Vivado Design Suite Tcl Command Reference Guide

UG835 (v2014.2) June 4, 2014

XILINX
ALL PROGRAMMABLE™

Revision History

Commands Deprecated as of 2014.2

- **export_hardware:** Use 'write_hwdef' for pre synthesis export and 'write_sysdef' after bitstream instead.
- **get_delays:** Use 'get_speed_models' instead.
- **get_gtbanks:** Use 'get_iobanks -filter {BANK_TYPE == BT_MGT}' instead.
- **open_netlist_design:** Use 'link_design' or 'open_run' instead.
- **read_vcd:**
- **report_default_switching_activity:** Use 'report_switching_activity' to report default static probability and toggle rates.
- **reset_default_switching_activity:** Use 'set_switching_activity' to set up default static probability and toggle rates.
- **set_default_switching_activity:** Use 'set_switching_activity' to set up default static probability and toggle rates.

Commands Added in 2014.2

get_ip_upgrade_results, write_hwdef, write_sysdef

Commands Modified in 2014.2

calc_config_time, commit_hw_mig, commit_hw_sysmon, create_hw_axi_txn, delete_hw_axi_txn, encrypt, generate_mem_files, get_hw_axi_txns, get_hw_migs, get_hw_sysmon_reg, get_hw_sysmons, ltrace, ptrace, read_bd, readback_hw_cfgmem, refresh_hw_axi, refresh_hw_mig, refresh_hw_sysmon, register_simulator, report_hw_axi_txn, reset_hw_axi, run_hw_axi, run_state_hw_jtag, runtest_hw_jtag, scan_dr_hw_jtag, scan_ir_hw_jtag, set_hw_sysmon_reg, update_files, write_bmm, write_inferred_xdc, write_mem_info

XILINX
ALL PROGRAMMABLE™

Chapter 1

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 2014.x** > **Vivado 2014.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 2014.x** > **Vivado 2014.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%/Roaming/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 discussd to meet your specific needs.

The `init.tcll` 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.



IMPORTANT: *Relative paths within the tcl.pre and tcl.post scripts are relative to the appropriate run directory of the project they are applied to: <project>/<project.runs>/<run_name>. You can use the DIRECTORY property of the current project or current run to define the relative paths in your Tcl hook scripts:*

```
get_property DIRECTORY [current_project]
get_property DIRECTORY [current_run]
```

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] [-verbose]
           [<patterns>]

Returns:
list of cell 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 cells of these pins, timing paths, nets, bels, sites
                   or drc violations
  [-match_style]   Style of pattern matching
                   Default: sdc
                   Values: ucf, sdc
  [-quiet]         Ignore command errors
  [-verbose]       Suspend message limits during command execution
  [<patterns>]     Match cell names against patterns
                   Default: *

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 are Vivado first-class objects, 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 the command 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_propert**y command:

```
report_property [get_cells inst_1]
```

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

Table 1–1: **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

Key	Value	Type
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 if 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.

Handling Lists of Objects

Commands that return more than one object, such as **get_cells** or **get_sites**, return a collection in the Vivado tool that looks and behaves like a native Tcl list. This feature allows performance gains when handling large lists of Tcl objects without the need to use special commands like the foreach_in_collection command. In the Vivado Design Suite collections can be processed like Tcl lists using built-in commands such as lsort, lsearch, and foreach.

Typically, when you run a **get_*** command, the returned results are echoed to the console and to the log file as a Tcl string, rather than as a list due to a feature of Tcl called "shimmering". Internally, Tcl can store a variable or value both as a string and as a faster native object such as a float or a list object. In Tcl, shimmering occurs when the representation of the object or value changes from the list object to the string object, or from string to list. A list of Vivado objects is returned by the **get_*** command, but the shimmered string representation is written to the log file and Tcl console.

However, to improve performance and prevent overloading memory buffers, the Vivado Design Suite limits and truncates the shimmered string to a default character length defined by the **tcl.collectionResultDisplayLimit** parameter, which has a default value of 500. Commands that can return a significant number of objects, such as **get_cells** or **get_sites**, will truncate the returned string, ending it with an ellipsis ('...'). You can use the **set_param** command to change the **tcl.collectionResultDisplayLimit** parameter value to return more or fewer results.



CAUTION! *The combination of shimmering and the **tcl.collectionResultDisplayLimit** parameter prevents the use of **in** and **ni** list operators in the Vivado Design Suite. Since a string shimmered from the list may be truncated, the **in** and **ni** operators cannot effectively determine if a specified object is in, or not-in, a list of objects. You should use list commands such as **lsearch** and **lsort** instead of **in** or **ni**.*

```
if {[lsearch -exact [get_cells *] $cellName] != -1} {...}
```

You can capture the complete list returned by the **get_*** command by assigning the results to a Tcl variable:

```
set allSites [get_sites]
```

The actual list in the variable assignment includes the complete result set, and is not truncated by the **tcl.collectionResultDisplayLimit** parameter. An example of this is seen in hierarchically querying all the cells in a design:

```
%set allCells [get_cells -hierarchical]
DataIn_pad_0_i_IBUF[0]_inst DataIn_pad_0_i_IBUF[1]_inst \
DataIn_pad_0_i_IBUF[2]_inst DataIn_pad_0_i_IBUF[3]_inst \
DataIn_pad_0_i_IBUF[4]_inst ...
%llength $allCells
42244
%lindex $allCells end
wbArbEngine/s4/next_reg
```

In the preceding example, the result of the hierarchical **get_cells** command was assigned to the **$allCells** variable. In appearance, the results are truncated. However, a check of the length of the list reports more than forty thousand cell objects, and the last index in the list returns an actual object, and not an ellipsis.



TIP: *If necessary, you can also use the **join** command, to join the list of objects returned by the **get_*** Tcl command, with a newline (\n), tab (\t), or a space (" "), to display the un-truncated list of objects:*

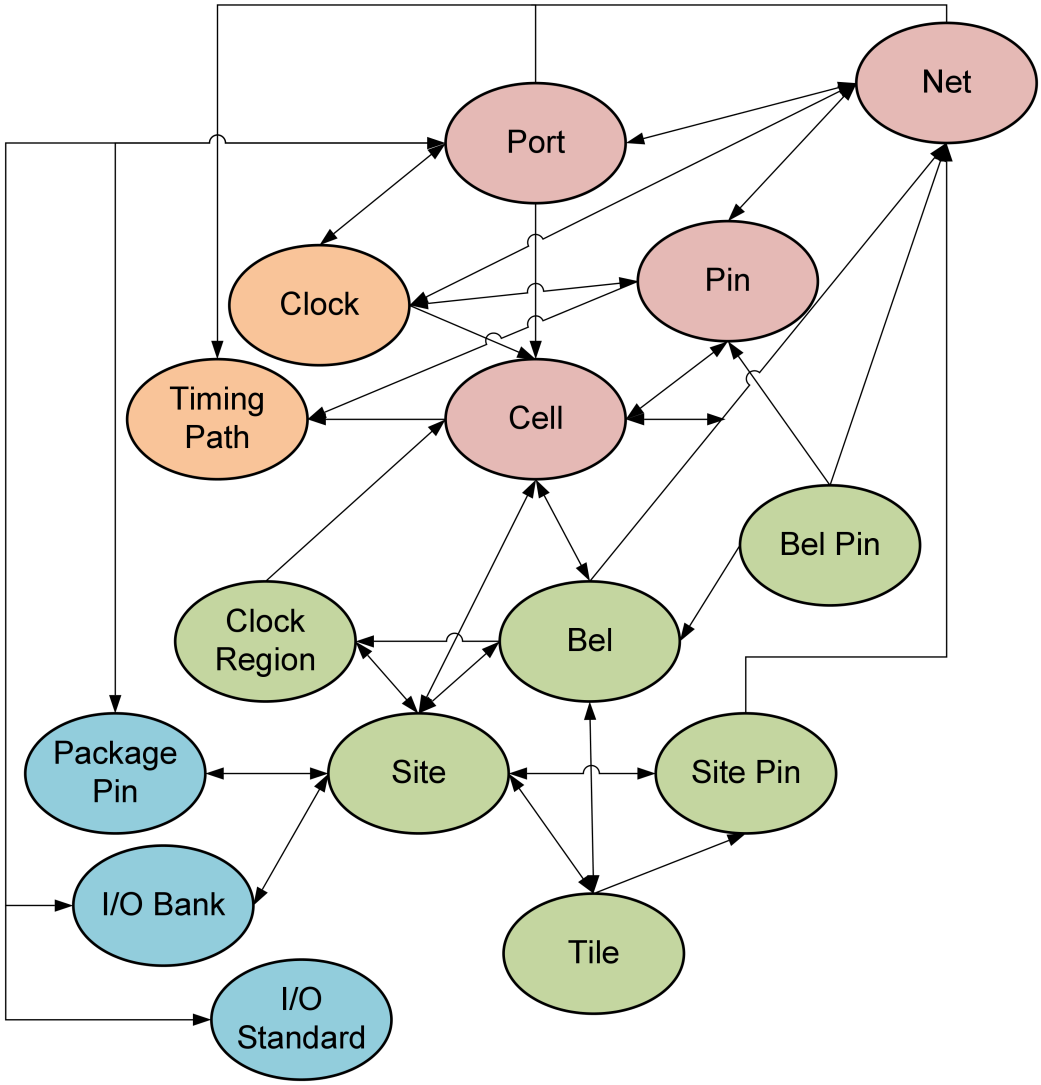
```
join [get_parts] " "
```

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. For more information on first class objects and their relationships, refer to the Vivado Design Suite Properties Reference Guide (UG912).



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

- Bitgen
- Board
- ChipScope
- CreatePeripheral
- DRC
- Debug
- FileIO
- Floorplan
- GUIControl
- Hardware
- IPFlow
- IPIntegrator
- Netlist
- Object
- PinPlanning
- Power
- Project
- PropertyAndParameter
- Report
- SDC
- Simulation
- SysGen
- Timing
- ToolLaunch
- Tools
- UpdateMem
- Waveform
- XDC
- XPS
- projutils
- user-written
- xilinxtclstore

Bitgen:

- calc_config_time

Board:

- apply_board_connection
- current_board
- current_board_part
- get_board_interfaces
- get_board_part_interfaces
- get_board_part_pins
- get_board_parts
- get_board_pins
- get_boards

ChipScope:

- launch_chipscope_analyzer
- launch_impact

CreatePeripheral:

- add_peripheral_interface
- create_peripheral
- generate_peripheral
- write_peripheral

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_violations
- remove_drc_checks
- report_drc
- reset_drc
- reset_drc_check

Debug:

- apply_hw_ila_trigger
- connect_debug_port
- create_debug_core
- create_debug_port
- delete_debug_core
- delete_debug_port
- disconnect_debug_port
- get_debug_cores
- get_debug_ports
- implement_debug_core
- report_debug_core
- write_debug_probes

FileIO:

- config_webtalk
- encrypt
- generate_mem_files
- infer_diff_pairs
- pr_verify
- read_bd
- read_checkpoint
- read_csv
- read_edif
- read_ip
- read_saif
- read_schematic
- read_twx
- read_verilog
- read_vhdl
- read_xdc
- write_bitstream
- write_bmm
- write_bsdl
- write_cfgmem
- write_checkpoint
- write_csv
- write_debug_probes
- write_edif
- write_ibis
- write_inferred_xdc
- write_mem_info
- write_schematic
- 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
- resize_pblock
- swap_locs
- unplace_cell

GUIControl:

- endgroup
- get_highlighted_objects
- get_marked_objects
- 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:

- apply_hw_ila_trigger
- boot_hw_device
- close_hw
- close_hw_target
- commit_hw_mig
- commit_hw_sio
- commit_hw_sysmon

- commit_hw_vio
- connect_hw_server
- create_hw_axi_txn
- create_hw_bitstream
- create_hw_cfgmem
- create_hw_sio_link
- create_hw_sio_linkgroup
- create_hw_sio_scan
- create_hw_sio_sweep
- current_hw_device
- current_hw_ila
- current_hw_ila_data
- current_hw_server
- current_hw_target
- delete_hw_axi_txn
- delete_hw_bitstream
- delete_hw_cfgmem
- detect_hw_sio_links
- disconnect_hw_server
- display_hw_ila_data
- display_hw_sio_scan
- get_cfgmem_parts
- get_hw_axi_txns
- get_hw_axis
- get_hw_devices
- get_hw_ila_datas
- get_hw_ilas
- get_hw_migs
- 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_sweeps
- get_hw_sio_txs
- get_hw_sysmon_reg
- get_hw_sysmons
- get_hw_targets
- get_hw_vios
- open_hw
- open_hw_target
- program_hw_cfgmem

- program_hw_devices
- read_hw_ila_data
- read_hw_sio_scan
- read_hw_sio_sweep
- readback_hw_cfgmem
- refresh_hw_axi
- refresh_hw_device
- refresh_hw_mig
- refresh_hw_server
- refresh_hw_sio
- refresh_hw_sysmon
- refresh_hw_target
- refresh_hw_vio
- remove_hw_sio_link
- remove_hw_sio_linkgroup
- remove_hw_sio_scan
- remove_hw_sio_sweep
- report_hw_axi_txn
- reset_hw_axi
- reset_hw_ila
- reset_hw_vio_activity
- reset_hw_vio_outputs
- run_hw_axi
- run_hw_ila
- run_hw_sio_scan
- run_hw_sio_sweep
- run_state_hw_jtag
- runtest_hw_jtag
- scan_dr_hw_jtag
- scan_ir_hw_jtag
- set_hw_sysmon_reg
- stop_hw_sio_scan
- stop_hw_sio_sweep
- upload_hw_ila_data
- wait_on_hw_ila
- wait_on_hw_sio_scan
- wait_on_hw_sio_sweep
- write_hw_ila_data
- write_hw_sio_scan
- write_hw_sio_sweep

IPFlow:

- add_peripheral_interface
- copy_ip
- create_ip
- create_ip_run
- create_peripheral
- delete_ip_run
- generate_peripheral
- generate_target
- get_ip_upgrade_results
- get_ipdefs
- get_ips
- import_ip
- open_example_project
- read_ip
- report_ip_status
- reset_target
- synth_ip
- update_ip_catalog
- upgrade_ip
- validate_ip
- write_peripheral

IPIntegrator:

- apply_bd_automation
- apply_board_connection
- 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
- read_bd
- regenerate_bd_layout
- replace_bd_cell
- save_bd_design
- ungroup_bd_cells
- upgrade_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

- apply_board_connection
- create_drc_check
- create_drc_ruledeck
- current_board
- current_board_part
- delete_drc_check
- delete_drc_ruledeck
- delete_hw_bitstream
- filter
- find_routing_path
- get_bel_pins
- get_bels
- get_board_interfaces
- get_board_part_interfaces
- get_board_part_pins
- get_board_parts
- get_board_pins
- get_boards
- get_cells
- get_cfgmem_parts
- get_clock_regions
- get_clocks
- get_debug_cores
- get_debug_ports
- get_designs
- get_drc_checks
- get_drc_ruledecks
- get_drc_violations
- get_files
- get_filesets
- get_generated_clocks
- get_highlighted_objects
- get_hw_axi_txns
- get_hw_axis
- get_hw_devices
- get_hw_ila_datas
- get_hw_ilas
- get_hw_migs
- 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_sweeps
- get_hw_sio_txs
- get_hw_sysmons
- get_hw_targets
- get_hw_vios
- get_interfaces
- get_io_standards
- get_iobanks
- get_ip_upgrade_results
- get_ipdefs
- get_ips
- get_lib_cells
- get_lib_pins
- get_libs
- get_macros
- get_marked_objects
- get_net_delays
- get_nets
- get_nodes
- get_package_pins
- get_parts
- get_path_groups
- get_pblocks
- get_pins
- get_pips
- get_pkgpin_bytegroups
- get_pkgpin_nibbles
- get_ports
- get_projects
- get_property
- get_runs
- get_selected_objects
- get_site_pins
- get_site_pips
- get_sites
- get_slrs
- get_speed_models
- 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

- run_state_hw_jtag
- runtest_hw_jtag
- scan_dr_hw_jtag
- scan_ir_hw_jtag
- 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
- report_power
- report_power_opt
- reset_operating_conditions
- reset_switching_activity
- set_operating_conditions
- set_power_opt
- set_switching_activity

Project:

- add_files
- add_peripheral_interface
- apply_board_connection
- archive_project
- close_design
- close_project
- copy_ip
- create_fileset
- create_ip_run
- create_peripheral

- create_project
- create_run
- create_xps
- current_board
- current_board_part
- current_fileset
- current_project
- current_run
- delete_fileset
- delete_ip_run
- delete_runs
- find_top
- generate_peripheral
- generate_target
- get_board_interfaces
- get_board_part_interfaces
- get_board_part_pins
- get_board_parts
- get_board_pins
- get_boards
- get_files
- get_filesets
- get_ip_upgrade_results
- 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_checkpoint
- 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
- synth_ip
- update_compile_order
- update_design
- update_files
- wait_on_run
- write_hwdef
- write_peripheral
- write_sysdef

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:

- calc_config_time
- check_timing
- create_drc_violation
- create_slack_histogram
- delete_clock_networks_results
- delete_timing_results
- delete_utilization_results
- get_msg_config
- 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_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_ram_utilization
- report_route_status
- report_seu
- report_ssn
- report_switching_activity
- report_timing
- report_timing_summary
- report_transformed_primitives
- report_utilization
- reset_drc
- reset_msg_config
- reset_msg_count
- reset_ssn
- reset_timing
- set_msg_config
- 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
- config_compile_simlib
- create_fileset
- current_scope
- current_sim
- current_time
- current_vcd
- delete_fileset
- describe
- flush_vcd
- generate_mem_files
- 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
- register_simulator
- 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
- create_slack_histogram
- delete_timing_results
- get_net_delays
- get_timing_arcs
- get_timing_paths
- report_clock_interaction
- report_clock_networks
- report_clocks
- report_config_timing
- report_datasheet
- report_disable_timing
- report_drc
- 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
- register_simulator

Tools:

- link_design
- list_features
- load_features
- opt_design
- phys_opt_design
- place_design
- route_design
- synth_design

UpdateMem:

- update_mem

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
- connect_debug_port
- create_clock
- create_debug_core
- create_debug_port
- create_generated_clock
- create_macro
- create_pblock
- create_property
- current_design
- current_instance
- delete_macros
- delete_pblock
- filter
- get_bel_pins
- get_bels
- get_cells
- get_clocks
- get_debug_cores
- get_debug_ports
- get_generated_clocks
- get_hierarchy_separator
- get_iobanks
- get_macros
- get_nets
- get_nodes
- get_package_pins
- get_path_groups
- get_pblocks
- get_pins
- get_pips
- get_pkgpin_bytegroups
- get_pkgpin_nibbles

- get_ports
- get_property
- get_site_pins
- get_site_pips
- get_sites
- get_slrs
- get_speed_models
- get_tiles
- get_timing_arcs
- get_wires
- 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_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:

- get_board_parts
- get_boards

projutils:

- export_simulation
- write_project_tcl

user-written:

- export_simulation
- write_project_tcl

xilinxtclstore:

- export_simulation
- write_project_tcl

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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file_name	Filename to add the breakpoint
line_number	Line number of the given file to set the breakpoint

Categories

Simulation

Description

The **add_bp** command lets you add breakpoints to an HDL source file to pause the current simulation.

A breakpoint is a user-determined stopping point in the source code used for debugging the design. When simulating a design with breakpoints, simulation of the design stops at each breakpoint to let you examine values and verify the design behavior.

You can report breakpoints in the current simulation using the **report_bps** command, and remove existing breakpoints using the **remove_bps** command.

This command returns a new breakpoint object if there is not already a breakpoint set at the specified file line, or returns an existing breakpoint object if there is already a breakpoint defined for the specified file and line number.



TIP: *You can capture the returned breakpoint object in a Tcl variable if needed*

The **add_bp** command returns an error if the command fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - (Required) The name of the HDL source file to add a breakpoint to.

line_number - (Required) The line number of the specified *file_name* to add the breakpoint to.

Examples

The following example adds a the breakpoint to the HDL source file at the specified line number:

```
add_bp C:/Data/ug937/sources/sinegen.vhd 137
```

See Also

- remove_bps
- report_bps

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 to the pblock only primitive cells from the specified list of cells.
[**-clear_locs**]	Clear instance location constraints
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
pblock	Pblock to add cells to
[*cells*]	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 in an open implemented design. 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 select all cells in a hierarchical module, and assign only the leaf-cells of the selected cells 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 only the leaf-cells 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*id* 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
condition_expression	The condition expression when true executes the given commands
commands	Commands to execute upon condition

Categories

Simulation

Description

Add a condition that is evaluated by a specified condition, condition_expression, and runs a series of simulation Tcl commands when the condition is TRUE.

Conditions can be defined prior to starting the simulation. When a condition is added, the simulator evaluates the condition expression anytime a signal change is detected. When a specified condition expression becomes TRUE, the condition commands are run.

The **add_condition** command returns a condition identifier for the added condition, or returns an error if the command fails.

Arguments

-name *arg* - (Optional) Provide a unique name for the condition. If no name is specified, then a default named is automatically created.

-radix *arg* - (Optional) Specifies the radix to use for the value of the condition. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

condition_expression - (Required) Specify an expression for the condition. If the condition evaluates to true, the simulation will run the specified *commands*. Specific operators that can be used in condition expressions are "equal" (==), and "not-equal" (!=). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

commands - (Required) Specify the Tcl commands or Tcl procedure to run when the *condition_expression* is true. This command is surrounded by {} (braces). The command can include standard Tcl commands and simulation Tcl commands, except **run**, **restart**, and **step**. Tcl variables used in the condition expression are surrounded by quotes "" instead of {} so variable substitution can be performed. Refer to the *Vivado Design Suite User Guide: Using Tcl Scripting (UG894)* for more information on variable substitution.

Examples

The following example defines a condition named resetLow, that becomes true when the reset signal is low, and then puts a message to the standard output, and stops the current simulation:

```
add_condition -name resetLow {/testbench/reset == 0 } {
puts "Condition Reset was encountered at [current_time]. Stopping simulation."
stop }
```

This next example defines a Tcl procedure, called myProc, that uses the **add_force** command to define `clk` and `reset` signal values, and print a standard message when it completes. A condition is then added that calls myProc when reset is low:

```
proc myProc {} {
  add_force clk {0 1} { 1 2} -repeat_every 4 -cancel_after 500
  add_force reset 1
  run 10 ns
  remove_force force2
  puts "Reached end of myProc"
}
add_condition -radix unsigned /top/reset==0 myproc
```

See Also

- add_force
- stop

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_check' 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_check' 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.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

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

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 -ruledeck 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_ruledeck placer+
add_drc_checks -of_objects [get_drc_ruledecks placer_checks] -ruledeck placer+
add_drc_checks -ruledeck 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} -ruledeck warn_only
```

See Also

- create_drc_check
- create_drc_ruledeck
- 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**] [**-copy_to** *arg*] [**-force**] [**-scan_for_includes**] [**-quiet**] [**-verbose**] [*files*...]

Returns

List of file objects that were added

Usage

Name	Description
[**-fileset**]	Fileset name
[**-norecurse**]	Do not recursively search in specified directories
[**-copy_to**]	Copy the file to the specified directory before adding it to project
[**-force**]	Overwrite the existing file when -copy_to is used
[**-scan_for_includes**]	Scan and add any included files found in the fileset's RTL sources
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*files*]	Name of the files and/or directories to add. Must be specified if -scan_for_includes 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.

IMPORTANT: *Adding multiple files one at a time can cause noticeable performance degradation. It is more efficient to use a single* ***add_files*** *command to import a list of files:*

```
add_files {file1 file2 file3 ... fileN}
```

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.

-copy_to *arg* - (Optional) Copy the selected files to the specified directory before adding the files to the project. This option lets you move files from their current location to a new folder to reference as part of the source structure for a project.

-force - (optional) Overwrite any existing files of the same name as the specified files, in the destination directory, when you use the **-copy_to** option.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 reads a top-level design netlist, and the char_fifo IP in a Non-Project Mode design:

```
# Read top-level EDIF and IP DCP
read_edif ./sources/wave_gen.edif
add_files ./my_IP/char_fifo/char_fifo.xci
```

NOTE: Either **add_files** or **read_ip** can be used reading in an IP core. All output products of the IP, including a design checkpoint (DCP), will be read as needed

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
- 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
[**-radix**]	Specifies which radix to use. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii
[**-repeat_every**]	Repeat every time duration
[**-cancel_after**]	Cancel after time offset
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hdl_object	Specifies the object upon which to add a force
values	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_peripheral_interface

Add a new bus interface to a peripheral.

Syntax

add_peripheral_interface -interface_mode *arg* **-axi_type** *arg* [**-quiet**] [**-verbose**] *name peripheral*

Returns

Nothing

Usage

Name	Description
-interface_mode	Mode of an interface, supported option - master,slave.
-axi_type	Type of a axi interface, supported option - lite,full,stream.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Name to initialize the newly added element e.g S1_AXI, M1_AXI
peripheral	Peripheral object

Categories

Project, IPFlow, CreatePeripheral

Description

Add an AXI bus interface to a peripheral created with the **create_peripheral** command.

Arguments

-interface_mode [master | slave] - (Optional) Specify the interface as a slave or master interface. The master interface generates out-bound AXI transactions and thus is the source of an AXI transfer. A slave interface receives in-bound AXI transactions and is the target of an AXI transfer.

-axi_type *arg* - (Optional) Type of AXI interface to add. The supported values are: full, lite, and stream.

- The full AXI4 interface is for memory mapped interfaces allowing bursts of up to 256 data transfer cycles with just a single address phase.
- The AXI4-Lite protocol is a subset of the AXI4 protocol intended for communication with simpler, smaller control register-style interfaces in components.
- The AXI4-Stream protocol is designed for unidirectional data transfers from master to slave with greatly reduced signal routing.

  NOTE: For more information on AXI interfaces refer to the *AXI Reference Guide (UG761)*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) User-specified name of the interface to add.

peripheral - (Required) The peripheral object to add the interface to. The peripheral is created with the **create_peripheral** command, which should be captured in a Tcl variable to facilitate further processing by this and other related commands. See the example below.

Example

This example creates a new AXI peripheral, with the VLNV attribute as specified, and captures the peripheral object in a Tcl variable for later processing, then adds AXI slave interfaces to the peripheral:

```
set perifObj [ create_peripheral {myCompany.com} {user} {testAXI1} {1.3} \
-dir {C:/Data/new_periph} ]
add_peripheral_interface {S0_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
add_peripheral_interface {S1_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
add_peripheral_interface {S2_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
```

See Also

- create_peripheral
- generate_peripheral
- write_peripheral

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**]	the wave configuration, group, or virtual bus into which the new wave object(s) will be inserted.
[**-at_wave**]	inserts the new wave object(s) into the specified wave object, or after the specified wave object if not a group or virtual bus
[**-after_wave**]	inserts the new wave objects(s) after the specified wave object
[**-before_wave**]	inserts the new wave objects(s) before the specified wave object
[**-reverse**]	reverses the displayed bit order of the new wave objects(s)
[**-radix**]	sets the displayed radix of the new wave object(s) to the specified radix. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii
[**-color**]	sets the displayed color of the new wave object(s) to the specified color, which can be a standard color name or a string of the form #RRGGBB
[**-name**]	sets the displayed name of the single new wave object to the specified string
[**-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**]	if the design object is a scope, this option specifies that wave objects for all design objects under that scope should be created
[**-regexp**]	interprets *items* using regular expressions
[**-nocase**]	only when regexp is used, performs a case insensitive match
[**-quiet**]	Ignore command errors

Name	Description
[**-verbose**]	Suspend message limits during command execution
items	the design objects from which to create 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/tcl8.5/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/tcl8.5/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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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**]	the wave configuration or group into which the new divider will be inserted.
[**-at_wave**]	inserts the new divider into the specified wave object, or after the specified wave object if not a group
[**-after_wave**]	inserts the new divider after the specified wave object
[**-before_wave**]	inserts the new divider before the specified wave object
[**-color**]	sets the displayed color of the new divider to the specified color, which can be a standard color name or a string of the form #RRGGBB Default: default
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*name*]	the displayed name of the new divider to the specified string 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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**]	the wave configuration or group into which the new group will be inserted.
[**-at_wave**]	inserts the new group into the specified wave object, or after the specified wave object if not a group
[**-after_wave**]	inserts the new group after the specified wave object
[**-before_wave**]	inserts the new group before the specified wave object
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*name*]	the displayed name of the new group to the specified string Default: new_group

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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**]	the wave configuration in which to create the marker
[**-name**]	sets the name of the new marker to the specified string
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*time*]	the numerical portion of the new marker's time Default: 0
[*unit*]	the time unit portion of the new marker's time. Allowed values are fs, ps, ns, us, ms, and s.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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**]	the wave configuration, group, or virtual bus into which the new virtual bus will be inserted.
[**-at_wave**]	inserts the new virtual bus into the specified wave object, or after the specified wave object if not a group or virtual bus
[**-after_wave**]	inserts the new virtual bus after the specified wave object
[**-before_wave**]	inserts the new virtual bus before the specified wave object
[**-reverse**]	reverses the displayed bit order of the new virtual bus
[**-radix**]	sets the displayed radix of the new virtual bus to the specified radix. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii
[**-color**]	sets the displayed color of the new virtual bus to the specified color, which can be a standard color name or a string of the form #RRGGBB Default: default
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*name*]	the displayed name of the new virtual bus to the specified string Default: new_virtual_bus

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	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, or use the **filter** command to filter the results returned by **all_clocks**.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- filter
- 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
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, and filters that list to return a single DSP assigned to the specified SITE:

```
filter [all_dsps] {SITE == DSP48_X1Y6}
```

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
[**-startpoints_only**]	Find only the timing startpoints
[**-flat**]	Hierarchy is ignored
[**-only_cells**]	Only cells
[**-levels**]	Maximum number of cell levels to traverse:Value >= 0 Default: 0
[**-pin_levels**]	Maximum number of pin levels to traverse:Value >= 0 Default: 0
[**-trace_arcs**]	Type of network arcs to trace: Values: timing, enabled, all
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
to	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-endpoints_only**]	Find only the timing endpoints
[**-flat**]	Hierarchy is ignored
[**-only_cells**]	Only cells
[**-levels**]	Maximum number of cell levels to traverse:Value >= 0 Default: 0
[**-pin_levels**]	Maximum number of pin levels to traverse:Value >= 0 Default: 0
[**-trace_arcs**]	Type of network arcs to trace: Values: timing, enabled, all
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
from	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
```

This example gets the fanout for all inputs assigned to I/O Bank 15 in the current design:

```
all_fanout [filter [all_inputs] {IOBANK == 15}]
```

See Also

- all_fanin
- current_instance
- filter
- 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

XDC

Description

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

You can use the **get_cells** command, or use the **filter** command to limit the results from the **all_ffs** command to return a list of flip-flop cells matching the specified properties.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
```

This example filters the results of **all_ffs** to return only the FDRE flops:

```
filter [all_ffs] {REF_NAME == FDRE}
```

See Also

- all_latches
- all_registers
- current_instance
- filter
- get_cells

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

To get a list of specific inputs in the design, use the **get_ports** command, or use the **filter** command to filter the results returned by **all_inputs**.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
```

This example gets all input port objects, filters out the GT ports, and sets the IOSTANDARD property for the non-GT ports:

```
set non_gt_ports [filter [all_inputs] {!is_gt_term}]
set_property IOSTANDARD LVCMOS18 $non_gt_ports
```

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
- filter
- get_clocks
- get_ports
- set_input_delay
- set_property

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
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

To get a list of specific outputs in the design, use the **get_ports** command, or use the **filter** command to filter the results returned by **all_outputs**.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- filter
- 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
[**-quiet**]	Ignore command errors
[**-verbose**]	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.

To get a list of specific RAM cells in the design, use the **filter** command to filter the results returned by **all_rams** based on properties assigned to the RAM cells.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 cells in the design:

```
all_rams
```

This example returns all RAM cells in the design, and filters the results to return only the FIFO block memories:

```
filter [all_rams] {PRIMITIVE_SUBGROUP == fifo}
```

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

The list of registers returned by **all_registers** can also be limited or reduced by the **filter** command to filter according to properties assigned to the registers.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 sets the minimum delay :

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

See Also

- all_clocks
- 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
objects	The objects to run the automation rule on

Categories

IPIntegrator

Description

IP Integrator provides the Designer Assistance feature, using the **apply_bd_automation** command, to automatically configure and/or add other relevant IP Integrator cells around a selected IP Integrator object. For more information on the Designer Assistance features refer to the *Vivado Design Suite User Guide: Designing with IP (UG896)* or the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994)*.

Currently block and connection automation exists for cells, interfaces, pins and ports. The Block Automation feature is provided for certain complex blocks such as the Zynq Processing System 7, MicroBlaze processor, AXI Ethernet and MIG memory controller.

The Connection Automation feature helps automate different types of connections. For instance when connecting Slave AXI-MM interfaces, the automation will also connect up the relevant clock and reset pins and also create an interconnect if one is required. Connection Automation may also help with board-level connections; connecting pins and interfaces from relevant cells, to external ports and interfaces, and applying appropriate board constraints on these external I/Os.

NOTE: This IP Integrator command is issued from within the Vivado IDE via the Designer Assistance GUI feature. It is recommended that you make use of this command in IP Integrator through the Vivado IDE, rather than directly from Tcl scripts. Use the **write_bd_tcl** command to output TCL for use within a user script.

Arguments

-config *args* - (Optional) Specify a list of configuration parameters for the IP Integrator object, and the value to assign to the parameter. The parameter name (param) is provided without quotes, while the value is quoted to distinguish it from the param. The parameter and value are assigned as a pair (param "value"). Multiple param "value" pairs can be defined for the specified *objects*, and should be enclosed in braces, {}.

```
-config {local_mem "16KB" ecc "Basic" debug_module "Debug Only"}
```

-rule *arg* - (Required) Specify the defined automation rules to use for the selected object.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Integrator object to apply automation to. The object must be returned by commands like **get_bd_cells**, get_bd_pins, and get_bd_interface, and cannot simply be referenced by name. Only a single IP Integrator object should be specified, and it must be compatible with the specified **-rule** and **-config** options.

Example

The following example applies Block Automation to the MicroBlaze cell according to the specified rules, configuring the specified parameters with the provided values:

```
apply_bd_automation -rule xilinx.com:bd_rule:microblaze \
   -config {local_mem "16KB" ecc "Basic" debug_module "Debug Only" \
   axi_periph "1" axi_intc "1" clk "New Clocking Wizard (100 MHz)" }  \
   [get_bd_cells /microblaze_1]
```

In this example, the Connection Automation feature applies the board rules to an IP subsystem pin or interface, in this case the clock interface object, when a known compatible interface is available on the board. The first command, **get_board_interfaces**, returns the interfaces on the target board that are compatible with the IP object. The second command, **apply_bd_automation**, connects the clock interface to the selected board interface:

```
get_board_interfaces -filter "VLNV==[get_property VLNV \
   [get_bd_intf_pins clk_wiz_1/CLK_IN1_D]]"
sys_diff_clock
apply_bd_automation -rule xilinx.com:bd_rule:board \
   -config {Board_Interface "sys_diff_clock" } \
   [get_bd_intf_pins /clk_wiz_1/CLK_IN1_D]
```

This example applies custom board rules to the IP subsystem clock interface object, CLK_IN1_D, when a clock interface is not available on the target board:

```
apply_bd_automation -rule xilinx.com:bd_rule:board \
   [get_bd_pins /clk_wiz_2/CLK_IN1_D]
```

This example applies custom board rules to an IP subsystem reset_pin, ext_reset_in, when the reset interface on the board is not available:

```
apply_bd_automation -rule xilinx.com:bd_rule:board \
   -config {rst_polarity "ACTIVE_HIGH" } \
   [get_bd_pins /proc_sys_reset_1/ext_reset_in]
```

See Also

- create_bd_cell
- create_bd_design
- get_bd_cells
- get_bd_intf_pins
- get_bd_pins
- get_board_interfaces
- write_bd_tcl

apply_board_connection

Applies board connections to given designs.

Syntax

```
apply_board_connection [-board_interface arg] -ip_intf arg
-diagram arg [-quiet] [-verbose]
```

Returns

Sucess/failure status of applied action

Usage

Name	Description
[**-board_interface**]	Board interface name to which connection need to be applied.
-ip_intf	Full path of IP interface name to which board automation need to be applied.
-diagram	The IP Integrator design name.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Object, Project, Board, IPIntegrator

Description

Connects the interface pin of an IP core in the specified block design to the interface of the current board part in the current project or design.

The board part provides a representation of the Xilinx FPGA device in the context of the board-level system, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces. The board part file stores information regarding board attributes. The file, called `board_part.xml`, is located in the `data/boards/board_parts` folder of the Vivado Design Suite installation area.

The command lets you quickly connect compatible interface pins of an IP Integrator block design to the appropriate interface definition on the current board part. To make the connection between the IP core to the board part, the IP Integrator feature of the Vivado Design Suite adds an external interface port and interface connection to the block design. The added external interface port is named for the specified board part interface.

The **apply_board_connection** commands uses the available interfaces of the current board part defined in the project. An error is returned if the project uses a target part rather than a target board. You can use the **current_board_part** command to identify the target board used by the project, or **get_board_parts** to list the boards available for use by the project. Use the **get_board_part_interfaces** command to determine the list of available interfaces on the current board.

To remove an existing IP interface connection, specify the **-ip_intf** option, but do not specify the **-board_interface**. If no board part interface is specified, the IP interface pin is disconnected.

This command returns a transcript of it actions, or returns an error if it fails.

Arguments

-board_interface *arg* - (Optional) Specifies the interface definition to connect to the specified IP core interface. If the **-board_interface** option is not specified, or is specified with an empty string value, "", an existing connection on the IP interface pin will be removed.

-ip_intf *arg* - (Required) Specifies the interface pin of an IP core in the IP Integrator Block Design defined by the **-diagram** option. The IP Interface pin is specified in the form "IP_core_name/interface_pin_name". In the first example below, the IP core instance name is `"/proc_sys_reset_0"` and the Interface pin is `"/ext_reset"`, combined as seen in the example.

-diagram *arg* - (Required) The name of the IP Integrator block design where the IP core instance is found.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 connects the SGMII interface, defined in the current board part, to the processor reset IP core:

```
apply_board_connection -board_interface "reset" \
-ip_intf "/proc_sys_reset_0/ext_reset" -diagram "design_2"
```

This example removes the connection on the specified interface pin, because the **-board_interface** option is an empty string:

```
apply_board_connection -board_interface "" -ip_intf "/proc_sys_reset_0/ext_reset" \
-diagram "design_2"
```

See Also

- create_port
- current_board_part
- get_board_part_interfaces
- get_board_parts
- write_checkpoint
- write_edif

apply_hw_ila_trigger

Apply trigger at startup init values to an ILA core in the design.

Syntax

apply_hw_ila_trigger [**-ila** *args*] [**-quiet**] [**-verbose**] [*file*]

Returns

Nothing

Usage

Name	Description
[**-ila**]	Apply trigger settings to ILA core at this ila cell
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*file*]	ILA startup trigger settings file

Categories

Hardware, Debug

Description

Apply a trigger configuration file to the bitstream of a design, to support ILA trigger at startup.

This command is used to configure the trigger settings of an ILA core in a design bitstream (.bit) file, so that the ILA debug core is armed to trigger on events immediately after device configuration and startup. This allows data to be captured from the earliest stages of device activity, which would not be possible through the use of the Hardware Manager feature of the Vivado Design Suite, and the **run_hw_ila** command.

The **apply_hw_ila_trigger** command reads a trigger configuration file written by **run_hw_ila -file** and applies the various trigger settings to the ILA core in the implemented design. The trigger configuration for the ILA core then become part of the bitstream written by **write_bitstream**, that is used to program the Xilinx FPGA device.

The process for using the trigger at startup feature includes the following steps:

1. From the Hardware Manager, use **run_hw_ila -file** to export the trigger register map file for the ILA core.
2. Open the implemented design, or the implemented design checkpoint.
3. Use the **apply_hw_ila_trigger** command to apply the trigger settings to the in-memory design.
4. Use the **write_bitstream** command to write the bitstream with the applied trigger configuration file.

   NOTE: Be sure to use the write_bitstream command, and not the Flow Navigator commands in the Vivado IDE
5. Return to the Hardware Manager, and use **program_hw_device** to program the hw_device using the new bitstream file.

Once programmed, the new ILA core should immediately arm at startup. In the Vivado logic analyzer feature, you should see the "Trigger Capture Status" for the ILA core is now populated with captured data samples if trigger events or capture conditions have occurred. Refer to the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)* for more information.

Arguments

-ila *arg* - (Optional) Apply trigger settings to the ILA core in the specified cell. The cell must be specified as an object using the **get_cells** command.

NOTE: The trigger configuration file includes the instance information for the ILA cell it applies to. The **-ila** option is not required, and should only be used to apply the file to a different ILA cell in the design

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) Specify a trigger configuration file that was created by the **run_hw_ila -file** command.

Example

The following example applies the trigger configurations to the ILA cell instance that is defined in the specified file:

```
apply_hw_ila_trigger C:/Data/ila1_triggers.tas
```

See Also

- current_hw_device
- current_hw_ila
- get_hw_ilas
- run_hw_ila
- write_bitstream

archive_project

Archive the current project.

Syntax

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

Returns

True

Usage

Name	Description
[**-force**]	Overwrite existing archived file
[**-exclude_run_results**]	Exclude run results from the archive
[**-include_config_settings**]	Include current project environment configuration settings/files in archive
[**-include_runs_in_progress**]	Include run result even if the run is in progress, this switch will be ignored if -exclude_run_results is specified
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*file*]	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.

-include_config_settings - (Optional) Add any initialization Tcl commands (`init.tcl`) to the archive file under the *project_name*`/config_settings` folder.

-include_runs_in_progress - (Optional) Include data in the archive from the directories of incomplete runs. This switch will be ignored if **-exclude_run_results** is specified.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-target_address_space**]	Target address space to place segment into
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	The objects to assign

Categories

IPIntegrator

Description

Assign unmapped IP address segments to address spaces in the IP Integrator subsystem design. If the target address space is not specified, the IP Integrator feature will automatically assign the address segment to an available address space.

This command returns the newly mapped address segments, or returns an error if it failed.

Arguments

-target_address_space *arg* - (Optional) The target address space to place the segment into.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - The address segment objects to assign to the target address space.

Example

The following example automatically assigns the specified address segment object to an address space. In this example, the target space is not provided, so the IP Integrator feature automatically assigns one:

```
assign_bd_address [get_bd_addr_segs \
{/microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB/Mem }]
```

See Also

- create_bd_addr_seg
- get_bd_addr_segs
- get_bd_addr_spaces

boot_hw_device

Issue JTAG Program command to hw_device.

Syntax

`boot_hw_device [-disable_done_check] [-quiet] [-verbose]` *`hw_device`*

Returns

Nothing

Usage

Name	Description
[**-disable_done_check**]	Disable done check for boot device
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_device	Target hw_device connection

Categories

Hardware

Description

Issue JTAG PROGRAM command to the hw_device (FPGA).

The boot_hw_device command triggers the FPGA boot and board startup sequence. The boot sequence starts the FPGA configuration process to clear the device of any prior programming, and then to load a new program, depending on the mode pin settings.

The hw_device will boot based on its mode pin settings. If the FPGA's mode pins on the device are set to JTAG mode, or the interface is not active (e.g. the PROM is not configured) the net effect of the boot_hw_device command is to clear the prior programming.

This command returns a 1 if it detects that the DONE pin has gone HIGH, or when the device has been cleared, otherwise it returns 0.

Arguments

-disable_done_check - (Optional) Disable done check for boot device. Use this option when you intend to clear a programmed device without reprogramming it, in JTAG mode for instance.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_device - (Optional) Specify the hw_device object to to send the JTAG boot command to. The hw_device must be specified as an object as returned by the **get_hw_devices** or **current_hw_device** commands.

Example

The following example creates a memory device and associates it with the **current hw_device**, assigning various properties of the hw_cfgmem object. Then it uses the **write_cfgmem** command to format the design bitstream, and associate the cfgmem (.mcs) file with the PROGRAM.FILE property of the hw_cfgmem object. Then it programs the hw_cfgmem object with the data from the MCS file starting at the specified offset address. After programming, the **boot_hw_device** command is issued for the **current_hw_device** and the DONE pin is checked, and an appropriate message is returned:

```
set memObj [create_hw_cfgmem -hw_device [current_hw_device] \
   [lindex [get_cfgmem_parts {28f00ap30t-bpi-x16}] 0]]
set_property PROGRAM.BLANK_CHECK  1 $memObj
set_property PROGRAM.ERASE  1 $memObj
set_property PROGRAM.CFG_PROGRAM  1 $memObj
set_property PROGRAM.VERIFY  1 $memObj
write_cfgmem -force -format MCS -size 64 -interface BPIx16 \
   -loadbit "up 0x0 ./project_netlist.runs/impl_1/sinegen_demo.bit" \
   ./config_28f00ap30t
set_property PROGRAM.FILE ./config_28f00ap30t.mcs $memObj
program_hw_cfgmem -offset 0x002000 -hw_cfgmem $memObj
if {[boot_hw_device [current_hw_device]] == 1} {
   puts stderr "DONE signal is HIGH"
} else {
   puts stderr "DONE signal is LOW"
}
```

See Also

- connect_hw_server
- create_hw_cfgmem
- current_hw_device
- current_hw_target
- get_hw_devices
- get_hw_targets
- program_hw_cfgmem

calc_config_time

Calculate device configuration time (ms).

Syntax

```
calc_config_time [-verbose] [-max] [-min] [-typ] [-por_used]
[-por_ramp arg] [-clk_freq arg] [-quiet]
```

Returns

Report

Usage

Name	Description
[**-verbose**]	Print out calculation parameters
[**-max**]	Calculate Maximum Configuration Time
[**-min**]	Calculate Minimum Configuration Time
[**-typ**]	Calculate Typical Configuration Time
[**-por_used**]	Specify if Power On Reset (POR) is used
[**-por_ramp**]	Specify a Power On Reset (POR) ramp rate as 1 ms to 50 ms Default: 0 ms
[**-clk_freq**]	Specify a clock frequency for Slave mode, or for Master mode if using external master clock (ms) Default: 0 ms
[**-quiet**]	Ignore command errors

Categories

Bitgen, Report

check_timing

Check the design for possible timing problems.

Syntax

```
check_timing [-file arg] [-no_header] [-loop_limit arg] [-append]
[-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)
[**-no_header**]	do not generate a report header
[**-loop_limit**]	Limit the number of loops reported for loops check Default: 100
[**-append**]	Append the results to file, don't overwrite the results file
[**-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 with the **-name** option, or to a file with **-file**.

Default Timing Checks:

- **constant_clock** - Checks for clock signals connected to a constant signal (gnd/vss/data).
- **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.
- **latch_loops** - Checks for and warns of combinational latch loops in the design.
- **loops** - Checks for and warns of combinational feedback loops in the design.
- **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.
- **no_clock** - Reports unclocked registers. In this case, no setup or hold checks are performed on data pins related to the register clock pin.
- **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 unclocked 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 unclocked will not be checked for output delays.
- **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: Unclocked 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: Unclocked output ports are not checked for partial output delays.
- **pulse_width_clock** - Reports clock pins that have only a pulse width check associated with the pin, and no setup or hold check, no recovery, removal, or clk->Q check.
- **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.
- **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.

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.

-append - Append the results to the specified file. As a default the **check_timing** command will overwrite an existing file when the **-file** argument is specified.

-no_header - (Optional) Do not write the standard header to the report. This is a boolean option that is enabled by its use.

-loop_limit *arg* - (Optional) The number of loops to identify and report during the **loop** and **latch_loop** checks. The **check_timing** command will continue to perform other checks after the **-loop_limit** has been reached.

-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 only the specified checks.



TIP: *Multiple checks should be enclosed in quotes, "", or braces, {}.*

-include *args* - (Optional) Run the specified checks in addition to the current default checks.

-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 default checks.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Description

Close the Hardware Manager tool in the Vivado Design Suite.

Opening the Hardware Manager using the **open_hw** command, is the first step in programming and/or debugging your design in Xilinx FPGA hardware. For more information refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)*.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 closes the Hardware Manager in the Vivado Design Suite:

```
close_hw
```

See Also

- connect_hw_server
- open_hw

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

Description

Close the connection to the current or specified hardware target that was previously opened using the **open_hw_target** command.

The hardware target is a system board containing a JTAG chain of one or more Xilinx FPGA devices that you can program with a bitstream file, or use to debug your design. Connections between hardware targets on the system board and the Vivado Design Suite are managed by the **vcse_server** application. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a list of supported JTAG download cables and devices.

This command returns connection messages from the hardware server, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_target - (Optional) Specify the hw_target object to close the connection to. The hw_target must be specified as an object as returned by the **get_hw_targets** or **current_hw_target** commands. If no target is specified, the open connection to the **current_hw_target** will be closed.

Example

The following example closes the current hardware target:

```
close_hw_target
```

See Also

- get_hw_targets
- open_hw_target
- refresh_hw_target

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*wcfgobj*]	Closes and destroys the specified wave configuration object, or the current wave configuration if none specified Default: NULL

Categories

Waveform

Description

Close the current, or specified wave configuration.

In the Vivado® simulator GUI, you can work with a waveform to analyze your design and debug your code. A wave configuration object displays with top-level HDL objects, and can be further populated using commands like **add_wave** and **add_wave_divider**. A new wave configuration object can be created in the current simulation with the **create_wave_config** command.

Any changes made to a wave configuration object can be saved to a Wave Config file with the **save_wave_config** command. You can open a saved Wave Config file with the **open_wave_config** command.

Arguments

-force - (Optional) Forces the closing of the Wave Config file, even if changes would be lost. The default behavior is to return an error if unsaved changes would be lost.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

wcfgobj - (Optional) Specify a Wave Config object to close. The default is to close the current Wave Config file as returned by the **current_wave_config** command.

NOTE: The wave configuration must be specified as an object using the **get_wave_configs** command

Examples

The following example closes all Wave Config files associated with the current simulation:

```
close_wave_config [get_wave_configs]
```

See Also

- create_wave_config
- current_wave_config
- get_wave_configs
- open_wave_config
- open_wave_database
- save_wave_config

commit_hw_mig

Commit the property changes of the current hardware object. Inputs can be any mig, device, target, or server hardware object. At least one object is required.

Syntax

`commit_hw_mig` [**`-quiet`**] [**`-verbose`**] *`hw_objects`*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_objects	hardware objects

Categories

Hardware

commit_hw_sio

Commit the property changes of the current hardware object. Inputs can be any serial I/O (except scan and sweep), device, target, or server 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
hw_objects	hardware objects

Categories

Hardware

Description

Commit the current values of properties defined on the specified serial I/O hardware objects in the Hardware Manager feature of the Vivado Design Suite to the current hardware device.

Specified objects can include any serial I/O object such as GTs, RXs, TXs, PLLs, or Commons, excluding hw_sio_scan and hw_sio_sweep objects. SIO objects also include device, target, or server hardware objects.

The SIO IBERT core operates on an object property-based set/commit use model. You can set the property values on the hardware objects using the **set_property** command. You can then drive those values onto the current hardware device using the **commit_hw_sio** command.



TIP: *To update the properties on the hardware object with the actual value on the device use the **refresh_hw_sio** command*

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_objects - (Required) Specify one or more hw_sio objects to refresh. The hw_sio objects must be specified as objects returned by on of the **get_hw_sio_*** commands.

Example

The following example sets the value of the CPLL_REFCLK_DIV property on the specified serial I/O GT object, and then commits the values of all properties on that object to the current hardware device:

```
set_property CPLL_REFCLK_DIV 3 [get_hw_sio_gts *MGT_X0Y8]
commit_hw_sio [list [get_hw_sio_gts *MGT_X0Y8}] ]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property
- set_property

commit_hw_sysmon

Commit the property changes of the current hardware object. Inputs can be hw_server, hw_target, hw_device or hw_sysmon objects. At least one object is required.

Syntax

`commit_hw_sysmon` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`hw_objects`*

Returns

Nothing

Usage

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

Categories

Hardware

Description

Commit the current values defined on the probes of the VIO Debug core to the current hardware device.

The Virtual Input/Output (VIO) debug core can both monitor and drive internal signals on a programmed Xilinx FPGA device in real time. The VIO core uses hardware probes, hw_probe objects, to monitor and drive signals on the device. Input probes monitor signals as inputs to the VIO core. Output probes drive signals to specified values from the VIO core.

The VIO core operates on an object property-based set/commit use model. You can set the OUTPUT_VALUE property on the output probes of the VIO core using the **set_property** command. You can then drive those values onto probed signals on the hardware device using the **commit_hw_vio** command.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_objects - (Optional) Specify the hw_vio debug core, or the hw_probe objects, to commit the OUTPUT_VALUE properties of onto the hw_device. You can commit the values of one or more hardware probes, or you can the values on all the probes of the VIO core by specifying the hw_vio object.

NOTE: The objects must be specified using the **get_hw_vios** or **get_hw_probes** commands, rather than specified by name

Example

The following example demonstrates the OUTPUT_VALUE property of the reset hw_probe being set high and committed to the hw_device, to trigger the reset process. Then the value is set low to release the reset:

```
set_property OUTPUT_VALUE 1 [get_hw_probes fast_cnt_reset -of_objects \
[get_hw_vios hw_vio_1]]
commit_hw_vio [get_hw_probes {fast_cnt_reset} -of_objects [get_hw_vios hw_vio_1]]
set_property OUTPUT_VALUE 0 [get_hw_probes fast_cnt_reset -of_objects \
[get_hw_vios hw_vio_1]]
commit_hw_vio [get_hw_vios hw_vio_1]
```

NOTE: In the first occurrence of the commit_hw_vio command, a single hardware probe is specified as the object, while the whole hw_vio debug core is specified in the second occurrence of the command

See Also

- connect_hw_server
- current_hw_device
- get_hw_probes
- get_hw_vios
- program_hw_devices
- refresh_hw_vio
- reset_hw_vio_activity
- reset_hw_vio_outputs
- set_property

compile_simlib

Compile simulation libraries.

Syntax

```
compile_simlib [-directory arg] [-family arg] [-force] [-language arg]
[-library arg] [-print_library_info arg] -simulator arg
[-simulator_exec_path arg] [-source_library_path arg] [-32bit] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-directory**]	Directory path for saving the compiled results Default: .
[**-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
[**-print_library_info**]	Print Pre-Compiled library information
-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
[**-32bit**]	Perform the 32-bit compilation
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Simulation

Description

Compile Xilinx® simulation libraries for use in the current project, or from a specified directory for use in all projects.



IMPORTANT: *The **compile_simlib** command should be rerun any time a new third party simulator, or a new Vivado Design Suite version or update is installed*

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 **compile_simlib** command uses simulator compilation directives when compiling the simulation libraries. You can edit the default configuration settings using the **config_compile_simlib** command.

The command returns information related to the compiled libraries, or an error if it fails.

Arguments

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

NOTE: By default, the libraries are saved in the current working directory in Non-Project mode, and the libraries are saved in "`<project>/<project>.cache/compile_simlib`" directory in Project mode. Refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)* for more information on Project and Non-Project modes

-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:

- all (generate libraries for all device families, the default)
- virtexu (for UltraScale™ Virtex® devices)
- virtex7 (for Virtex-7)
- kintexu (for UltraScale Kintex® devices)
- kintex7 (for Kintex-7)
- kintex7l (for Kintex-7 Lower Power)
- artix7 (for Artix®-7)
- artix7l (for Atix-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 simulation library to compile. As a default, the **compile_simlib** command will compile all simulation libraries. Valid values are:

- all (the default)
- unisim
- simprim

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

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

-print_library_info - (Optional) Print the library information for the compiled simulation library.

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

- **modelsim** - Version 10.2a and later
- **questa** - Version 10.2a and later
- **ies** - (Linux only) Version 13.20.005 or later
- **vcs_mx** - (Linux only) Version I-2014.03 or later
- **riviera** - Version 2013.06 or later
- **active_hdl** - (Windows only) Version 9.2 sp1

-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 or $XILINX_PLANAHEAD).

NOTE: Do not use this option unless explicitly instructed to by Xilinx® Technical Support.

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

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

- config_compile_simlib
- launch_modelsim

config_compile_simlib

Report info of simulation libraries.

Syntax

config_compile_simlib [**-cfgopt** *arg*] [**-simulator** *arg*] [**-reset**] [**-quiet**] [**-verbose**]

Returns

Nothing

Usage

Name	Description
[**-cfgopt**]	Configuration option in form of simulator.language.library.options
[**-simulator**]	Display the configurations for specified simulator
[**-reset**]	Reset all configurations
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Simulation

Description

Configure third party simulator options for use by the **compile_simlib** command.

The Vivado Design Suite has a pre-defined configuration file for the **compile_simlib** command, with compilation options defined for supported third-party simulators, languages, and libraries. The **config_compile_simlib** command is provided to let you change the configuration options for specific combinations of simulator, language, and library.

Use the **config_compile_simlib** command without any arguments to return all current configuration options.

Arguments

-cfgopt *arg* - (Optional) Specify the configuration options for a specific third party simulator, language, and library combination. The **-cfgopt** argument is specified as a string made up of four parts in the form:

{*simulator.language.library.options*}

Where:

- *Simulator* - Specify the third-party simulator to configure options for. Refer to the compile_simlib command for currently supported versions of third-party simulators. Valid values are:
  - **modelsim**
  - **questa**
  - **ies**
  - **vcs_mx**
  - **riviera**
  - **active_hdl**

- *Language* - Specify the language to set the simulation options for. Valid values are verilog or vhdl.
- *Library* - Specify the library to compile. Valid values for library are:
  - axi_bfm
  - ieee
  - simprim
  - std
  - unisim
  - vl

- Options - Configuration options specific to the simulator, language, and library specified. The following are the default compilation options available for different *Simulator.Language.Library* combinations:
  - Active HDL: -v2k5 (verilog), +define+XIL_TIMING, -93 (vhdl), -nowarn ELAB1_0026 (vhdl),
  - Incisive Enterprise Simulator: -MESSAGES, -NOLOG, -DEFINE XIL_TIMING, -v93 (vhdl), -RELAX (vhdl)
  - ModelSim: -novopt, -quiet, +define+XIL_TIMING, -93 (vhdl), -source (simprim, unisim)
  - Questa: -novopt, -quiet, +define+XIL_TIMING, -93 (vhdl), -source (simprim, unisim)
  - Riviera: -v2k5 (verilog), +define+XIL_TIMING, -93 (vhdl), -nowarn ELAB1_0026 (vhdl)
  - VCS MX: -sverilog (verilog), -nc, +v2k (simprim, unisim), +define+XIL_TIMING

  NOTE: Refer to the third-party simulator documentation for other compilation options that may be supported

-simulator *arg* - (Optional) This option acts as a filter to return only the configuration options associated with the specified simulator. Refer to the **-cfgopt** option for a list of valid simulator values.

-reset - (Optional) Restore all settings to the default configuration options of the Vivado Design Suite.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 compilation options for the Modelsim simulator, Verilog language, and Unisim library:

```
config_compile_simlib -cfgopt {modelsim.verilog.unisim: -quiet}
```

The following example configures the compilation options for multiple simulation libraries:

```
config_compile_simlib -cfgopt {modelsim.verilog.synopsys: -quiet} \
-cfgopt {modelsim.verilog.simprim:-source +define+XIL_TIMING}
```

See Also

compile_simlib

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
[**-enable_input_delay_default _clock**]	Launch SDC unclocked input delays from an internally defined clock: Values: true, false; This option is not supported for UCF constraints
[**-enable_preset_clear_arcs**]	Time paths through asynchronous preset or clear timing arcs: true, false;
[**-disable_flight_delays**]	Disable adding package times to IO Calculations : Values: true, false;
[**-quiet**]	Ignore command errors
[**-verbose**]	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 unclocked 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-info**]	Show whether WebTalk is currently enabled or disabled
[**-user**]	Enable/disable WebTalk for the current user. Specify either 'on' to enable or 'off' to disable. Default: empty
[**-install**]	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
[**-quiet**]	Ignore command errors
[**-verbose**]	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 dependent 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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: [Coretcl-120] Webtalk has been disabled by the current user.
INFO: [Coretcl-123] Webtalk has been enabled for the current installation.
INFO: [Coretcl-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*] [**-boundary_type** *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
[**-boundary_type**]	Used when source object is on a hierarchical block's interface pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. Default: both
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
object1	Name of intf_port or intf_pin to connect
object2	Name of intf_port or intf_pin to connect

Categories

IPIntegrator

Description

Connect the interface pins on an IP Integrator cell to other interface pins, or to external interface ports. An interface is a grouping of signals that share a common function in the IP Integrator subsystem design.

This command will create an interface net of the name specified by the **-intf_net** option, will connect to an existing interface net of the specified name, or will assign a name if none is specified.

Returns the connected interface net object, or returns an error.

Arguments

-intf_net *arg* - (Optional) Specifies the name of an existing interface net, previously created by the **create_bd_intf_net** command, or a new interface net that will be created. If no name is provided, the IP Integrator feature will automatically name the net.

-boundary_type [lower | upper | both] - (Optional) Specifies the search area for making a connection to an interface pin of an hierarchical block. The default of **both** searches the current level of the block design hierarchy, and downward, to find connecting objects. Specify **upper** to search only the current level of hierarchy, specify **lower** to search from the bottom of the hierarchy upward.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

object1 - (Required) The first interface pin or port to connect the net to.

object2 - (Required) The second interface pin or port to connect the net to.

Example

The following example connects an interface pin on an IP Integrator core to an interface port in the subsystem design:

```
connect_bd_intf_net [get_bd_intf_pins clk_wiz_1/CLK_IN1_D] \
[get_bd_intf_ports /diff_clock_rtl]
```

See Also

- create_bd_cell
- create_bd_intf_net

connect_bd_net

Connect port and pin object list.

Syntax

connect_bd_net [**-net** *arg*] [**-boundary_type** *arg*] [**-quiet**] [**-verbose**] *objects*...

Returns

0 if successful, error otherwise

Usage

Name	Description
[**-net**]	The single net that all objects connect to
[**-boundary_type**]	Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. Default: both
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	The objects connect to the net

Categories

IPIntegrator

Description

Create a new net in the current IP Integrator subsystem design connecting the specified list of block diagram port and pin objects, or connect an existing net to the specified pins and ports.

If the **-net** option is not specified, a new net is created connecting the listed objects. If **-net** is used, the specified net is either connected or created as needed.

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

You can us this command to connect pins or ports at different levels of the subsystem design hierarchy. However, in this case, you cannot specify the **-net** option because the connection, when complete, will result in multiple nets rather than a single net.

The command returns the connected IP Integrator subsystem design net object, or returns an error.

Arguments

-net *arg* - (Optional) Create a single net in the current IP subsystem design.

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.

-boundary_type [lower | upper | both] - (Optional) Specifies the search area for making a connection to a pin of an hierarchical block. The default of **both** searches the current level of the block design hierarchy, and downward, to find connecting objects. Specify **upper** to search only the current level of hierarchy, specify **lower** to search from the bottom of the hierarchy upward.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 port and pin objects in the current IP Integrator subsystem design.

Examples

The following example connects two pins on different levels of the IP subsystem design hierarchy:

```
connect_bd_net [get_bd_pins /vidOut_1/locked] [get_bd_pins /newMod1/bridge_1/fid]
```

NOTE: Because `/vidOut_1/locked` and `/newMod1/bridge_1/fid` are in different levels of the subsystem design hierarchy, the **-net** option is not specified. In this case, multiple nets are created for connection across the hierarchy.

See Also

- create_bd_net
- disconnect_bd_net
- 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
[**-channel_start_index**]	Connect nets starting at channel index
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
port	Debug port name
nets	List of nets or pins

Categories

Debug, XDC

Description

Connects a signal from the Netlist design to a port on an ILA 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 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 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.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

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.

Examples

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

```
create_debug_port myCore PROBE
set_property PORT_WIDTH 8 [get_debug_ports myCore/PROBE1]
connect_debug_port myCore/PROBE1 [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*] [**-url** *arg*] [**-quiet**] [**-verbose**]

Returns

Hardware server

Usage

Name	Description
[**-host**]	server host name Default: localhost
[**-port**]	server port number Default: 60001
[**-url**]	hw_server url Default: localhost:3121
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Hardware

Description



IMPORTANT: *You must use the **open_hw** command to open the Hardware Manager in the Vivado Design Suite before using this command.*

To open a hardware target containing a JTAG chain of one or more Xilinx FPGA devices, for programming and/or debugging your design, you must first connect to a Vivado tools hardware server (hw_server) and Vivado CSE server (vcse_server) to manage the connection to the hardware target (hw_target):

- **hw_server** - Manages the connection to the physical programming target. It should be running on the machine connected to the hardware programmer, or test board, connected either locally or remotely. The **hw_server** command must be launched as a separate application, and can be found in the `/bin` folder of your Vivado Design Suite installation directory.
- **vcse_server** - Running on the same machine as the Vivado Design Suite to manage the connection to the hw_server. The **vcse_server** is started automatically by the **connect_hw_server** command.

To connect to a hardware server, the **hw_server** application must be running, and the host name and port number noted for the **-url** argument of the **connect_hw_server** command. The default URL for the hw_server process is **localhost:3121**.

Open a connection to a Vivado Design Suite hardware server by specifying the host name and port number of the **vcse_server** application. The default host is localhost, and the default port is 6001. The Vivado tool will launch the **vcse_server** automatically on the local machine (localhost) if it is not already running. For more information on setting up and running the Vivado CSE server, refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)*.

You can connect a single instance of the Vivado Design Suite to multiple hardware servers, to support programming and debugging different device configurations. However, you can only have one connection to a specific hardware server as identified by the host name/port number combination. An error is returned if you attempt to open a connection to a server that is already connected.

The last connected hardware server is the current hardware server, unless changed by the **current_hw_server** command. Any connected server can be disconnected with the **disconnect_hw_server** command.

This command returns the host name of the hardware server that has been connected.

Arguments

-host *arg* - (Optional) The host name of the computer system running the **vcse_server** application. The default is **localhost**.

-url *arg* - (Optional) The URL of the running **hw_server application**. The URL consists of the *hostname:port_number*. The default setting is **localhost:3121**.

-port *arg* - (Optional) The port assigned to the **vcse_server** when the process was started. The default port is 6001.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 connects to the hw_server on the remote host trumpet3, through port 3121. The vcse_server is launched locally using port 6099:

```
connect_hw_server -host localhost -port 6099 -url trumpet3:3121
```

This example connects to the **vcse_server** on the localhost, using the default port of 60001, and connects to a running hw_server that is also running locally with the default url:

```
connect_hw_server
```

NOTE: If the **vcse_server** is not running on the localhost, it will be started by the Vivado Design Suite prior to making connection, however, the **hw_server** command must be run separately prior to connection

See Also

- current_hw_server
- disconnect_hw_server
- get_hw_servers
- get_hw_targets
- open_hw
- refresh_hw_server

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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 [list 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
parent_cell	Parent cell
objects	The objects to copy

Categories

IPIntegrator

Description

Use this command to copy IP Integrator objects from one open subsystem design to a second subsystem design. The selected objects can be copied into the top-level of the current subsystem design, or into an existing hierarchical module.

Because **get_bd_cells**, and other commands like it, operate on the current subsystem design, you must store the objects to be copied in a Tcl variable, as shown in the example below. Set the current subsystem design to the source design, select the group of objects to be copied, and store them in a Tcl variable. Then change the **current_bd_design** to the target design, and copy the selected objects. In this case, the **-from_design** option must be used.

You can also use this command to copy objects from one level of hierarchy in the current subsystem design to another. In this case, the **-from_design** argument does not need to be specified.

This command returns 0 if successful, and returns an error if it failed.

Arguments

-prefix *arg* - (Optional) A prefix name to apply to any cells that are copied into the hierarchical module.

-from_design *arg* - (Optional) The name of the IP Integrator subsystem design where the specified objects are located. The design must be open in IP Integrator. If **-from_design** is not specified, the objects are located in the current design, as determined by the **current_bd_design** command.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

parent_cell - (Required) The name of the hierarchical module to copy the specified objects into. You can specify "/" for the top-level of the current subsystem design.

objects - (Required) The list of cells and nets specified by the **get_bd_cells** and **get_bd_nets** commands, to copy into the specified *parent_cell*.

Example

The following example sets the current subsystem design, selects a group of objects, and stores them in a Tcl variable. The current subsystem design is then changed, and the selected objects are copied into the top-level of the current design:

```
current_bd_design myDesign
myDesign
set copyObjs [get_bd_cells {vidOut_1 bridge_1}]
/vidOut_1 /bridge_1
current_bd_design design_1
design_1
copy_bd_objs -from_design myDesign / $copyObjs
0
```

NOTE: Because the **get_bd_cells** command only returns cells from the current subsystem design, the Tcl variable is used to store those objects prior to switching to the target design with **current_bd_design**

See Also

- current_bd_design
- get_bd_cells
- get_bd_nets

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
objects	IP to be copied

Categories

Project, IPFlow

Description

Create a copy of an IP core that has been previously instanced into the current project.

Arguments

-name *arg* - (Required) Specify the name of the new IP to be created.

-dir *arg* - (Optional) The path to a directory outside of the local project to store the newly created IP. The specified directory must already exist, or the command will return an error.

NOTE: If a directory is not specified, the new IP will be added to the local project directory structure at *project_name*`.srcs/sources_1/ip`.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - (Required) The IP object to copy. The **copy_ip** command can only be used to copy a single IP core at one time. The IP must be specified as an IP object returned by the get_ips command, and not simply referenced by name.

Example

The following example create a copy of the FIFO core previously instanced into the current project and writes it to the specified directory:

```
copy_ip -name newFIFO -dir C:/Data/new_IP [get_ips oldFIFO]
```

See Also

- create_ip
- get_ips
- import_ip
- read_ip

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

Description

Create a new address segment object in the current IP Integrator subsystem design.

This command returns the newly created segment object, or returns nothing if it failed.

Arguments

-range *arg* - (Required) Range, or size, of address segment to create expressed as an integer or hexadecimal value. The range should be specified as a number of bits expressed as a power of 2, such as 4096, or as an amount of memory to allocate such as 4K, 16M, 1G.

-offset *arg* - (Required) Offset of the address segment. Written as an integer or hexadecimal value, such as 0x00000000.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

parent_addr_space - (Required) Defines the parent address space of the segment.

slave_segment - (Required) Defines a slave segment of the master segment being created.

name - (Required) The name of the address segment to create.

Example

The following example creates address segments for the Data and Instruction address spaces of the Microblaze core:

```
create_bd_addr_seg -range 0x10000 -offset 0x41200000 \
[get_bd_addr_spaces microblaze_1/Data] \
[get_bd_addr_segs microblaze_1_axi_intc/s_axi/Reg] SEG1
create_bd_addr_seg -range 0x40000000 -offset 0x0 \
[get_bd_addr_spaces microblaze_1/Instruction] \
[get_bd_addr_segs microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB/Mem] SEG1
```

See Also

- get_bd_addr_segs
- get_bd_addr_spaces

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
name	Name of cell to create

Categories

IPIntegrator

Description

Add a cell from the IP Integrator catalog to the current subsystem design, or create a new hierarchical module to add to the subsystem design.

When adding an IP core from the catalog, the **-vlnv** argument is required.

When creating a new hierarchical module, the **-type hier** argument is required.

This command returns the name of the newly created cell object, or returns nothing if the command fails.

Arguments

-vlnv *arg* - (Required) This option is required for **-type IP** (the default), or optional for **-type hier**. Specify the Vendor:Library:Name:Version attribute of the cell to add from the IP Integrator catalog. The VLNV attribute identifies the object in the IP Integrator catalog. This argument is not needed when creating a new hierarchical module.

NOTE: The **-vlnv** property for IP from the IP Integrator catalog refers to files in the Vivado Design Suite installation hierarchy that can be found at `data/ip/xilinx`.

-type [IP | hier] - (Optional) Specify the cell as being either IP or a hierarchical module. The default is to create an IP cell from the IP catalog.

NOTE: Although both **-vlnv** and **-type** are marked as Optional, one or the other must be specified. Use **-vlnv** to identify the IP core to add from the IP Integrator catalog, or specify **-type hier** to indicate the creation of a new hierarchical module

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 cell or hierarchical module to add to the current IP subsystem design.

Example

The following example adds an AXI FIFO core from the IP Integrator catalog to the current subsystem design, with the specified name:

```
create_bd_cell -vlnv xilinx.com:ip:axi_fifo_mm_s:4.0 axi_fifo_1
```

NOTE: The **-vlnv** argument identifies the core to add from the IP Integrator catalog.

This example creates a new hierarchical module, myModule1, and moves the AXI FIFO from the prior example into the new module. myModule1 is set as the current instance in the subsystem design, and a new module is created, myModule2, which is added to the current instance. Finally the current instance is reset to point to the top-level of the subsystem design:

```
create_bd_cell -type hier myModule1
/myModule1
move_bd_cells /myModule1 [get_bd_cells /axi_fifo_1]
/myModule1
current_bd_instance /myModule1
/myModule1
create_bd_cell -type hier myModule2
/myModule1/myModule2
current_bd_instance
/
```

See Also

- copy_bd_objs
- current_bd_instance
- move_bd_cells

create_bd_design

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

Syntax

`create_bd_design` [**`-dir`** *`arg`*] [**`-quiet`**] [**`-verbose`**] *`name`*

Returns

The newly created design object, "" if failed

Usage

Name	Description
[**-dir**]	Directory path for remote BD to be created and managed outside the project
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	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

-dir *arg* - (Optional) Specify the directory to write the block design file to. This lets you create and manage BD files outside of the current project directory structure, and facilitates reuse of block designs across multiple projects.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	Name of intf_net to create

Categories

IPIntegrator

Description

Create a new IP Integrator interface net for the subsystem design.

This command returns the newly created interface net object if successful, and returns noting if it failed.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 names of interface net to create.

Example

The following example creates an interface net in the current subsystem design:

```
create_bd_intf_net diff_clock_rtl
```

See Also

connect_bd_intf_net

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
name	Name of intf_pin to create

Categories

IPIntegrator

Description

Create a new interface pin on an IP Integrator hierarchical module. An IP Integrator interface is a grouping of signals that share a common function, and can include both individual signals and buses that share a related function. An AXI4-Lite master, for example, is an interface that includes a large number of individual signals plus multiple buses.

To create a single connection pin, or standard bus pin, use the **create_bd_pin** command.

Interface pins connect with other compatible interface pins, or interface ports. The interface pin is added as a port inside the hierarchical module, to connect outside of the module, and as a pin on the hierarchical module.

You must define the hierarchical module as the current instance in the IP Integrator subsystem design, using the **current_bd_instance** command. The current instance is the target of the **create_bd_intf_pin** command.

This command returns the name of the newly created interface pin object if successful, and returns an error if it failed.

Arguments

-vlnv *arg* - (Required) The Vendor:Library:Name:Version (VLNV) attribute of the interface pin object to be added to the subsystem design. The VLNV attribute identifies the object in the IP Integrator catalog.

NOTE: The **-vlnv** property for interface pins and ports refers to files in the Vivado Design Suite installation hierarchy. **-vlnv xilinx.com:interface:lmb_rtl:1.0** for example, is located in the Vivado Design Suite installation at `data/ip/interfaces/lmb_v1_0`.

-mode *arg* - (Required) Defines the mode of the interface pin. Accepted values are Master, Slave, System, MirroredMaster, MirroredSlave, MirroredSystem, Monitor.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 interface pin to add to the current instance.

Example

The following example sets the hierarchical module, newMod1, as the current instance of the IP integrator subsystem design, and then creates a new interface pin on that module:

```
current_bd_instance [get_bd_cells /newMod1]
create_bd_intf_pin -mode Slave -vlnv xilinx.com:user:dma_rtl:1.0 data_in
```

See Also

- connect_bd_intf_net
- create_bd_intf_pin
- create_bd_port
- get_bd_intf_ports

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
name	Name of port to create

Categories

IPIntegrator

Description

Create a new IP Integrator subsystem design interface port. An IP Integrator interface is a grouping of signals that share a common function, and can include both individual signals and buses that share a related function. An AXI4-Lite master, for example, is an interface that includes a large number of individual signals plus multiple buses.

To create a single connection port, or common bus port, use the **create_bd_port** command.

This command returns the name of the newly created interface port object if successful, and returns nothing if it failed.

Arguments

-vlnv *arg* - (Required) The Vendor:Library:Name:Version (VLNV) attribute of the interface port object to be added to the subsystem design. The VLNV attribute identifies the object in the IP Integrator catalog.

NOTE: The **-vlnv** property for interface pins and ports refers to files in the Vivado Design Suite installation hierarchy that can be found at . **-vlnv xilinx.com:interface:lmb_rtl:1.0** for example, is located in the Vivado Design Suite installation at `data/ip/interfaces/lmb_v1_0`.

-mode *arg* - (Required) Defines the mode of the interface pin. Accepted values are Master, Slave, System, MirroredMaster, MirroredSlave, MirroredSystem, Monitor.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 interface port to add to the subsystem design.

Example

The following example creates a new IP Integrator interface port and adds it to the current subsystem design:

```
create_bd_intf_port -vlnv xilinx.com:interface:diff_clock_rtl:1.0 \
-mode Slave diff_clock_rtl
```

See Also

- connect_bd_intf_net
- create_bd_intf_pin
- create_bd_port
- get_bd_intf_ports

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
name	Name of net to create

Categories

IPIntegrator

Description

Create a new net in the current IP Integrator subsystem design.

This command returns the newly created net object, or returns an error if failed.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 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 net:

```
create_bd_net net1
```

See Also

- connect_bd_intf_net
- connect_bd_net
- create_bd_cell
- create_bd_intf_net
- create_bd_intf_pin
- create_bd_intf_port
- create_bd_pin
- create_bd_port
- current_bd_design

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
name	Name of pin to create

Categories

IPIntegrator

Description

Create a new pin to add to an IP Integrator hierarchical module.

This command returns the name of the newly created pin object, or returns an error message if it failed.

Arguments

-from *arg* - (Optional) The starting index of a standard bus pin. This is unspecified for single bit pins.

-to *arg* - (Optional) The ending index of a standard bus pin. This is unspecified for single bit pins.

-dir [I | O | IO] - (Required) The direction of the pin. Valid values are I for input, O for output, and IO for bidirectional pins.

-type *arg* - (Optional) Defines the type of the pin as a clock pin (CLK), a reset pin (RST), a clock enable pin (CE), an interrupt pin (INTR), or as a data pin (DATA). If you do not define the pin type, it will be undefined (UNDEF).

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 subsystem pin to create. The name is referenced from the hierarchical module that the pin is being added to: **/modName/pinname**.

Examples

The following example creates a new input pin on the specified module in the current IP Integrator subsystem design:

```
create_bd_pin -dir I -type rst /newMod1/rst
/newMod1/rst
```

See Also

- create_bd_cell
- create_bd_intf_pin
- create_bd_intf_port
- create_bd_port

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
name	Name of port to create

Categories

IPIntegrator

Description

Create a new port to add to an IP Integrator subsystem design. The port is a connection to signals external to the subsystem design.

This command returns the name of the newly created port object, or returns an error message if it failed.

Arguments

-from *arg* - (Optional) The starting index of a standard bus port. This is unspecified for single bit ports.

-to *arg* - (Optional) The ending index of a standard bus port. This is unspecified for single bit ports.

-dir [I | O | IO] - (Required) The direction of the port. Valid values are I for input, O for output, and IO for bidirectional ports.

-type *arg* - (Optional) Defines the type of the port as a clock port (CLK), a reset port (RST), a clock enable port (CE), an interrupt port (INTR), or as a data port (DATA). If you do not define the port type, it will be undefined (UNDEF).

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 subsystem port to create.

Examples

The following example creates a new bidirectional bus port in the current IP Integrator subsystem design:

```
create_bd_port -from 0 -to 32 -dir IO -type data addr
/addr
```

See Also

- create_bd_cell
- create_bd_intf_pin
- create_bd_intf_port
- create_bd_port

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
cells	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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 myCell1 myCell2 myCell3
```

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 defined in nanoseconds (ns). 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.



IMPORTANT: *If you use **create_clock** to create a generated clock, instead of **create_generated_clock**, the created clock does not inherit any of the properties of its source clock. The insertion delay and jitter of the parent clock will not be propagated to the generated clock, causing incorrect timing calculations*

The **create_clock** 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).

-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 rising and falling 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 bftClk]
```

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 bftClk]
```

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

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
name	Name of the new debug core instance
type	Type of the new debug core

Categories

Debug, XDC

Description

Adds a new Integrated Logic Analyzer (ILA) debug core to an open Netlist Design in the current project. The ILA debug core defines ports for connecting nets to for debugging the design in the logic analyzer feature of the Vivado Design Suite available through the **open_hw** command.

ILA debug cores can be added to the RTL source files of the design, or added to the netlist after synthesis. Refer to the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)* for more information on using ILA debug cores.

NOTE: A debug core can only be added to an open Netlist Design using this command.

The ILA core is created with a CLK port and a PROBE port by default. The CLK port defines the clock domain for the ILA core, and allows you to probe signals that are common to that domain. The CLK port only supports one clock signal, and so you must create a separate debug core for each clock domain. The PROBE port provides a probe point for nets marked for debug with the MARK_DEBUG property. The PROBE port offers multiple channels to probe multiple nets from a single ILA core.

You can add new ports to an existing ILA 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 ILA 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.

NOTE: When the ILA core is added to the project, the tool also adds a Debug Hub core (labtools_xsdbmasterlib) 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 synthesis run, creating the specified netlist design name, and then creates a new ILA debug core in that design:

```
open_run -name netlist_1 synth_1
create_debug_core myCore ila
```

The properties of the debug core can be customized by using the **set_property** command as in the following example:

```
set_property C_DATA_DEPTH 2048 [get_debug_cores myCore]
```

This example marks a sequence of nets for debugging using the MARK_DEBUG property, creates a new debug core, connects the CLK port to the appropriate clock domain, and assigns the debug nets to the PROBE ports on the core:

```
set_property MARK_DEBUG true [get_nets [list {control_reg[0]} {control_reg[1]} \
{control_reg[2]} {control_reg[3]} {control_reg[4]} {control_reg[5]} \
{control_reg[6]} {control_reg[7]} {control_reg[8]} {control_reg[9]} \
{control_reg[10]} {control_reg[11]} {control_reg[12]} {control_reg[13]} \
{control_reg[14]} {control_reg[15]} {control_reg[16]} {control_reg[17]} \
{control_reg[18]} {control_reg[19]} {control_reg[20]} {control_reg[21]} \
{control_reg[22]} {control_reg[23]} {control_reg[24]} {control_reg[25]} \
{control_reg[26]} {control_reg[27]} {control_reg[28]} {control_reg[29]} \
{control_reg[30]} {control_reg[31]}]]
create_debug_core u_ila_0 ila
set_property port_width 1 [get_debug_ports u_ila_0/CLK]
connect_debug_port u_ila_0/CLK [get_nets [list wbClk ]]
set_property port_width 32 [get_debug_ports u_ila_0/PROBE0]
connect_debug_port u_ila_0/PROBE0 [get_nets [list {control_reg[0]} {control_reg[1]}\
 {control_reg[2]} {control_reg[3]} {control_reg[4]} {control_reg[5]} \
{control_reg[6]} {control_reg[7]} {control_reg[8]} {control_reg[9]} \
{control_reg[10]} {control_reg[11]} {control_reg[12]} {control_reg[13]} \
{control_reg[14]} {control_reg[15]} {control_reg[16]} {control_reg[17]} \
{control_reg[18]} {control_reg[19]} {control_reg[20]} {control_reg[21]} \
{control_reg[22]} {control_reg[23]} {control_reg[24]} {control_reg[25]} \
{control_reg[26]} {control_reg[27]} {control_reg[28]} {control_reg[29]} \
{control_reg[30]} {control_reg[31]} ]]
```

See Also

- connect_debug_port
- create_debug_port
- delete_debug_core
- get_debug_cores
- implement_debug_core
- open_hw
- open_run
- report_debug_core
- report_property
- set_property

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
name	Name of the debug core instance
type	Type of the new debug port

Categories

Debug, XDC

Description

Defines a new port to be added to an existing Vivado ILA debug core. The port provides connection points on an ILA core to attach nets from the design for debugging.

When a new debug core is created using the **create_debug_core** command, it includes a CLK and PROBE port by default. However, you can add trigger input/output port types as well. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on port types and purpose.

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 PROBE ports to support multiple signals using the **set_property port_width** command (see Examples).

NOTE: CLK, TRIG_IN, TRIG_IN_ACK, TRIG_OUT, and TRIG_OUT_ACK 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_core**.

type - (Required) The type of debug port to insert. The supported port types are:

- CLK - Defines the clock port for connecting the ILA debug core to a clock domain. Each debug core can only have one CLK port, and each CLK port can only connect to one clock domain. Therefore you must use multiple ILA cores to probe signals from different clock domains.
- PROBE - Provides probe points to connect to signals that are marked for debugging with the MARK_DEBUG property. The ILA debug core can contain multiple PROBE ports, which are automatically numbered by the Vivado tool when the port is added to the core. Each PROBE port can contain one or more channels, or connection points, as defined by the PORT_WIDTH property.
- TRIG_IN/TRIG_IN_ACK, and TRIG_OUT/TRIG_OUT_ACK - The ILA probe trigger comparators used to detect specific comparison conditions on the PROBE inputs to the ILA core. TRIG_IN and TRIG_IN_ACK, and TRIG_OUT and TRIG_OUT_ACK should be added to the debug core as port pairs when used. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information.

Examples

The following example creates a new debug core, and then adds an additional PROBE port to the core, then sets the width of that new port to 8, and connects signals to the PROBE port:

```
create_debug_core myCore ila
create_debug_port myCore PROBE
set_property PORT_WIDTH 8 myCore/PROBE1
connect_debug_port -channel_start_index 1 myCore/PROBE1 \
{m1_cyc_i m1_ack_o m1_err_o m1_rty_o}
```

NOTE: Recall that the ILA core is created with a CLK and PROBE port by default, so the new PROBE port is automatically numbered as PROBE1

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 [-hiername arg] -name arg [-desc arg] [-msg arg]
-rule_body arg [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-hiername**]	Specify the hiername for this rule. When the DRC UI panel is created, this is used to place the new rule in the menu hierarchy. Use a dot (.) to separate layers in the menu hierarchy. 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 checks 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 hierarchy 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 and 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

-hiername *arg* - (Optional) Defines a rule grouping for the new 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" hierarchy 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hierarchy name 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} -hiername {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} \
                 -hiername {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_violations
- 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
ruledecks	Names of DRC 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 the 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_violations** command. The design objects associated with a DRC violation object can be obtained using the **-of_objects** option of the appropriate **get_*** command, such as **get_cells**, **get_nets**, or **get_ports** for instance:

```
get_ports -of_objects [get_drc_violations -name drc_1 NSTD*]
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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} \
                 -hiername {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_cells
- get_drc_checks
- get_drc_violations
- get_nets
- get_pins
- get_ports
- get_sites
- 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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
[-invert] [-edge_shift args] [-add] [-master_clock arg] [-quiet]
[-verbose] objects
```

Returns

New clock object

Usage

Name	Description
[**-name**]	Generated clock name
[**-source**]	Master clock source object pin/port
[**-edges**]	Edge Specification
[**-divide_by**]	Period division factor: Value >= 1 Default: 1
[**-multiply_by**]	Period multiplication factor: Value >= 1 Default: 1
[**-combinational**]	Create a divide_by 1 clock through combinational logic
[**-duty_cycle**]	Duty cycle for period multiplication: Range: 0.0 to 100.0 Default: 0.0
[**-invert**]	Invert the signal
[**-edge_shift**]	Edge shift specification
[**-add**]	Add to the existing clock in source_objects
[**-master_clock**]	Use this clock if multiple clocks present at master pin
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	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.

Clocks can be added to a design in one of three ways:

- Primary physical or virtual clocks defined with the **create_clock** command.
- Derived clocks defined with the **create_generated_clock** command generated from a primary physical clock.
- Derived clocks automatically generated by the Vivado Design Suite when a clock propagates to an MMCM/PLL/BUFR.

You can also use the **create_generated_clock** command to change the name of clocks that the Vivado tool has auto-derived from an MMCM/PLL/BUFR. In this case, a new clock is not created, but an existing clock defined on the specified source object is renamed to the provided name. This requires **-name** and *object* to be specified, and supports the use of **-source** and/or **-master_clock** to further identify the clock to rename.



IMPORTANT: *You cannot rename a clock that is already in use by other constraints at the time of renaming*

This command returns the name of the clock object that is created, or returns an error if it fails.

Arguments

-name *arg* - (Optional) The name of the generated clock to create on the specified object, or the name to assign to an existing clock 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.

-invert - (Optional) Create a generated clock with the phase inverted from the master clock.

-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 object 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
```

The following example renames the automatically named clock that is derived by the Vivado Design Suite on the MMCM clock output:

```
create_generated_clock -name CLK_DIV2 [get_pins mmcm/CLKOUT1]
```

See Also

- check_timing
- create_clock
- get_generated_clocks
- get_pins
- set_clock_latency
- set_clock_uncertainty
- set_propagated_clock

create_hw_axi_txn

Create hardware AXI transaction object.

Syntax

```
create_hw_axi_txn [-address arg] [-data arg] [-size arg]
-type arg [-len arg] [-burst arg] [-cache arg] [-id arg] [-quiet]
[-verbose] name hw_axi
```

Returns

New hardware AXI transaction object

Usage

Name	Description
[**-address**]	AXI read or write address. Default: Address zero
[**-data**]	Transaction data. Default: All zeroes
[**-size**]	Burst size. Default: 32
-type	READ or WRITE transaction.
[**-len**]	Burst length. Default: 1
[**-burst**]	Burst type: INCR,FIXED or WRAPPED. Default: INCR
[**-cache**]	Cache type. Default: 3
[**-id**]	Address ID. Default: 0
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Name of new object.
hw_axi	Associated hardware AXI core object.

Categories

Hardware

create_hw_bitstream

Read bitstream file into memory.

Syntax

`create_hw_bitstream -hw_device` *`arg`* **`[-quiet] [-verbose]`** *`file`*

Returns

Nothing

Usage

Name	Description
-hw_device	Target hw_device connection
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Bitstream filename

Categories

Hardware

Description

Read a bitstream file, created with the **write_bitstream** command, to create a hw_bitstream object, and associate that object with a hw_device object in the Hardware Manager feature of the Vivado Design Suite.

The hw_bitstream object is associated with the specified hw_device through the PROGRAM.HW_BITSTREAM property on the device. This property is automatically set by the **create_hw_bitstream** command. The PROGRAM.FILE property is also set to reflect the file path of the specified bitstream file.

NOTE: A hw_bitstream object is also automatically created and associated with a hw_device object when you use the **program_hw_devices** command

The created hw_bitstream object can be removed with the **delete_hw_bitstream** command.

This command returns the name of the hw_bitstream file read, or returns an error if it fails.

Arguments

-hw_device *arg* - (Required) Specify the hw_device object to associate the hw_bitstream object with. The hw_device must be specified as an object as returned by the **get_hw_devices** or **current_hw_device** commands.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 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.

Example

The following example creates a hw_bitstream object from the specified bitstream file, and associates it with the current hw_device object:

```
create_hw_bitstream -hw_device [current_hw_device] C:/Data/design1.bit
```

See Also

- current_hw_device
- delete_hw_bitstream
- get_hw_devices
- program_hw_devices
- set_property
- write_bitstream

create_hw_cfgmem

Read cfgmem file into memory.

Syntax

`create_hw_cfgmem -hw_device` *`arg`* **`[-quiet] [-verbose]`** *`mem_device`*

Returns

Nothing

Usage

Name	Description
-hw_device	Target hw_device connection
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
mem_device	memory device name

Categories

Hardware

Description

Create a hw_cfgmem object associated with the specified hw_device.

The process whereby the bitstream data is loaded or programmed into the Xilinx® FPGA is called configuration. Configuration is designed to be flexible to accommodate different application needs and, wherever possible, to leverage existing system resources to minimize system costs.

Xilinx FPGAs are configured by loading design-specific configuration data, in the form of a bitstream file, into the internal memory of the hw_device. The hw_cfgmem defines a flash memory device used for configuring and booting the Xilinx FPGA device. Once the hw_cfgmem object is created, and associated with the hw_device, the configuration memory can be programmed with the bitstream and other data using the **program_hw_cfgmem** command.

The hw_cfgmem object is associated with the specified hw_device object through the PROGRAM.HW_CFGMEM property on the device object. To work with the hw_cfgmem object, use the get_property command to obtain the object from the hw_device:

```
get_property PROGRAM.HW_CFGMEM [current_hw_device]
```



TIP: *When creating a new hw_cfgmem object, you can also associate the object with a Tcl variable as shown in the example below. By referring to the variable, you can set properties on the object, and use the object with other Tcl commands like* ***program_hw_cfgmem*** *or* ***readback_hw_cfgmem***

This command returns the created hw_cfgmem object, or returns an error if it fails.

Arguments

-hw_device *arg* - (Optional) Specify the hw_device object to associate the memory configuration device with. The hw_device must be specified as an object as returned by the **get_hw_devices** command or **current_hw_device** command.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

mem_device - (Required) Specify the flash memory device to use for configuring the associated hw_device. The configuration memory must be specified as a cfgmem_part object object using the **get_cfgmem_parts** command.

NOTE: The COMPATIBLE_PARTS property of the cfgmem_part object identifies which Xilinx FPGA devices a configuration memory is compatible with

Example

The following example:

1. Gets the PART associated with the **current_hw_device**.
2. Gets a list of cfgmem_parts that are compatible with the device part and has more than a specified amount of memory.
3. Uses one of the compatible parts to create a new hw_cfgmem object.
4. Sets the PROGRAM.FILE property to the configuration memory file created from the bitstream file with the **write_cfgmem** command, that will be used to program the flash memory device.

```
set devPart [get_property PART [current_hw_device]]
set cfgParts [get_cfgmem_parts -filter \
   "COMPATIBLE_PARTS =~ [concat "{*$devPart*}"] && MEM_DENSITY > 64 "]
set cfgMem [create_hw_cfgmem -hw_device [current_hw_device] [lindex $cfgParts 0 ]]
set_property PROGRAM.FILE {C:/Data/cfgmem_file.mcs} $cfgMem
program_hw_cfgmem -hw_cfgmem $cfgMem
```

See Also

- current_hw_device
- delete_hw_cfgmem
- get_cfgmem_parts
- get_property
- program_hw_cfgmem
- readback_hw_cfgmem
- write_cfgmem

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 [**-description** *arg*] [**-quiet**] [**-verbose**] [*hw_sio_rx*] [*hw_sio_tx*]

Returns

The new hardware SIO link

Usage

Name	Description
[**-description**]	Description of link. Default: Link object name
[**-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

Description

Define a communication links between transmitter (TX) and receiver (RX) objects on the GTs of the IBERT debug core implemented on the current hardware device.

Vivado Serial I/O analyzer is a link-based analyzer, which lets you link between any transmitter and receiver within the IBERT design. The links define the communication paths and protocols between transmitters and receivers of the GigaBit transceivers on the device. You can configure the links by using the set_property command to specify property values on the link object. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on configuring links.

This command returns the created hw_sio_link object, or returns an error if it fails.

Arguments

-description *arg* - (Optional) Provide a brief description that acts as a label for the link. This description is stored on the DESCRIPTION property of the hw_sio_link object.



TIP: *The NAME property of the hw_sio_link objects is a full path designation of the link from the transmitter (TX) to receiver (RX) GTs on the IBERT debug core, including the hw_server, hw_target, hw_device, and hw_sio_ibert objects. Any description you provide here can be used as a shortcut to help locate the link.*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_rx - (Optional) Specify the receiver element of the GT to include in the communication link. The receiver must be specified as an object as returned by the **get_hw_rxs** command.

hw_sio_tx - (Optional) Specify the transmitter element of the GT to include in the communication link. The transmitter must be specified as an object as returned by the **get_hw_txs** command.



IMPORTANT: *Although the receiver and transmitter are marked as optional, at least one must be provided to create a link. A link with only a transmitter or receiver is considered an open-ended link*

Example

The following example creates a communication link between the specifed RX and TX

```
create_hw_sio_link -description Link_12  [get_hw_sio_txs *MGT_X0Y12*] \
[get_hw_sio_rxs *MGT_X0Y12*]
```



TIP: *In the example above the TX precedes the RX. The order of these objects is not significant*

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups

create_hw_sio_linkgroup

Create a new hardware SIO link group.

Syntax

```
create_hw_sio_linkgroup [-description arg] [-quiet]
[-verbose] hw_sio_links
```

Returns

The new hardware SIO link group

Usage

Name	Description
[**-description**]	Description of link group. Default: Link group object name
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_sio_links	hardware SIO links

Categories

Hardware

Description

Create a new group to associate the specified TX to RX communication links on the IBERT debug core implemented on the current device.

Vivado Serial I/O analyzer is a link-based analyzer. The links define the communication paths and protocols between transmitters and receivers of the GigaBit transceivers on the device. Link groups, or hw_sio_linkgroup objects, let you associate links into related groups, to collectively configure properties and run scans.

This command returns the name of the linkgroup created, or returns an error if the command fails.

Arguments

-description *arg* - (Optional) Provide a brief description that acts as a label for the link group. This description is stored on the DESCRIPTION property of the hw_sio_linkgroup object.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_links - (Required) Specify one or more hw_sio_link objects to associate together as a group. The hw_sio_links must be specified as objects as returned by the **create_hw_sio_link** or **get_hw_sio_links** commands.

NOTE: If you include a hw_sio_link object that already belongs to another linkgroup, the command will return an error

Example

The following example uses the DESCRIPTION property on hw_sio_link objects to identify which links to add to a new link group:

```
create_hw_sio_linkgroup -description "LoopBack Links" \
[get_hw_sio_links -filter {DESCRIPTION == Link_12 || DESCRIPTION == "Link 9" \
|| DESCRIPTION == "Link 10" || DESCRIPTION == "Link 11" }]
```

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups

create_hw_sio_scan

Create a new hardware SIO scan. If a Link object is passed in, it must have a RX Endpoint object.

Syntax

create_hw_sio_scan [**-description** *arg*] [**-link_settings** *arg*] [**-quiet**] [**-verbose**] *scan_type* [*hw_sio_object*]

Returns

The new hardware SIO scan

Usage

Name	Description
[**-description**]	Description of scan Default: Scan object name
[**-link_settings**]	List of Link properties and values to set before running the scan. Default: None
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
scan_type	Scan Type Options: 1d_bathtub, 2d_full_eye
[*hw_sio_object*]	RX endpoint or Link object to perform scan on. Default: None

Categories

Hardware

Description

Create a serial I/O analyzer scan object for the specified communication link on the IBERT debug core.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized Eye Scan hardware of Xilinx UltraScale devices or 7 Series FPGAs. The Vivado serial I/O analyzer feature lets you to create, run, and save link scans.

This command creates and returns a link scan object that you can use with the **run_hw_sio_scan** command to run analysis on the specified links, or GT receivers. You can also save the scan to disk using the **write_hw_sio_scan** command.

You can remove the created scan object using **remove_hw_sio_scan**.

This command returns the hw_sio_scan object, or returns an error if he command fails.

Arguments

-description *arg* - (Optional) Provide a brief description that acts as a label for the serial I/O analyzer scan. The description can be used to identify the hw_sio_scan object.

-link_settings *arg* - (Optional) Specify a list of Link properties and values to set before running the scan. If no link settings are provided, the default settings are used. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a description of scan properties and settings.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

scan_type - (Required) Specify the scan type. Valid types include:

- 1d_bathtub - Scan all horizontal sampling points through the 0 vertical axis.
- 2d_full_eye - Scan all horizontal and vertical sampling points to create an "eye".



TIP: *The results of the bathtub scan can be saved to a file with write_hw__sio_scan, but the plot cannot be displayed in the Vivado serial I/O analyzer using display_hw_sio_scan*

hw_sio_object - (Required) Specify the IBERT debug core link, hw_sio_link, or receiver, hw_sio_rx, to define the scan object for. The link or receiver must be specified as objects as returned by the **get_hw_sio_links** or **get_hw_sio_rxs** commands.

NOTE: The create_hw_sio_scan command requires the hw_sio_object to be specified as a list of one object

Example

The following example defines a scan for the specified link:

```
create_hw_sio_scan -description {LoopBack} 2d_full_eye [get_hw_sio_links Link_1]
```

See Also

- create_hw_sio_scan
- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_scans
- get_hw_sio_sweeps
- remove_hw_sio_scan
- run_hw_sio_scan
- stop_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

create_hw_sio_sweep

Create a new hardware SIO sweep. If a Link object is passed in, it must have a RX Endpoint object.

Syntax

```
create_hw_sio_sweep [-description arg] [-iteration_settings arg]
[-quiet] [-verbose] scan_type [hw_sio_link]
```

Returns

The new hardware SIO sweep

Usage

Name	Description
[**-description**]	Description of sweep Default: Sweep object name
[**-iteration_settings**]	List of LINK_SETTINGS for each scan to set before running the sweep. Default: None
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
scan_type	Sweep Type Options: 1d_bathtub, 2d_full_eye
[*hw_sio_link*]	Link object to perform sweep on. Default: None

Categories

Hardware

Description

Create a serial I/O analyzer link sweep object to run multiple scans across a range of values.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized features of Xilinx UltraScale devices or 7 Series FPGAs. It can also be helpful to run multiple scans on a the link with different configuration settings for the GTs. This can help you determine which settings are best for your design. The Vivado serial I/O analyzer feature enables you to define, run, and save link sweeps, or collections of link scans run across a range of values.

This command creates and returns a link sweep object that you can use with the **run_hw_sio_sweep** command to run analysis on the specified links, or GT receivers. You can also save the sweep scan to disk using the **write_hw_sio_sweep** command.

You can remove the created scan object using **remove_hw_sio_sweep**.

This command returns the hw_sio_sweep object, or returns an error if the command fails.

Arguments

-description *arg* - (Optional) Provide a brief description that acts as a label for the serial I/O analyzer sweep scan.

-iteration_settings *arg* - (Optional) Specify a list of properties to vary across multiple scans. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a description of iteration settings.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

scan_type - (Required) Specify the scan type. Valid types include:

- 1d_bathtub - Scan all horizontal sampling points through the 0 vertical axis.
- 2d_full_eye - Scan all horizontal and vertical sampling points to create an "eye".



TIP: *The results of the bathtub scan can be saved to a file with write_hw__sio_scan, but the plot cannot be displayed in the Vivado serial I/O analyzer using display_hw_sio_scan*

hw_sio_link - (Required) Specify the hw_sio_link to define the sweep object for. The link must be specified as objects as returned by the **get_hw_sio_links** command.

Example

The following example defines a varable for the hw_sio_sweep object created, then runs the sweep scan:

```
set xil_newSweep [create_hw_sio_sweep -description {Sweep 0} 2d_full_eye \
   [lindex [get_hw_sio_links *MGT_X0Y10/TX*] 0 ]]
run_hw_sio_sweep [get_hw_sio_sweeps $xil_newSweep]
```

See Also

- create_hw_sio_scan
- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_scans
- get_hw_sio_sweeps
- remove_hw_sio_scan
- remove_hw_sio_sweep
- run_hw_sio_sweep
- stop_hw_sio_sweep
- wait_on_hw_sio_sweep
- write_hw_sio_sweep

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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-force]
[-vendor arg] [-library arg] [-name arg] [-version arg] [-quiet]
[-verbose]
```

Returns

List of file objects that were added

Usage

Name	Description
[**-vlnv**]	VLNV string for the Catalog IP from which the new IP will be created (colon delimited Vendor, Library, Name, Version)
-module_name	Name for the new IP that will be added to the project
[**-dir**]	Directory path for remote IP to be created and managed outside the project
[**-force**]	Overwrite existing IP instance; allowed only with -dir option
[**-vendor**]	IP Vendor name
[**-library**]	IP Library name
[**-name**]	IP Name
[**-version**]	IP Version
[**-quiet**]	Ignore command errors
[**-verbose**]	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 VLNV string for the existing Catalog IP from which the new IP will be created. The VLNV is the *Vendor:Library:Name:Version* string which identifies the IP in the catalog. The VLNV 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 created. The module is created with the *module_name/module_name*`.xci` naming convention.

-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.

-force - (Optional) Overwrite an existing IP instance of the same *module_name*, if one exists in the specified directory. All files associated with the existing IP will be overwritten by files associated with the newly created IP. This option can only be used with the **-dir** option.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 CONFIG.COMPONENT_NAME  c_addsub_v11_0_0 [get_ips c_addsub_v11_0_0]
set_property CONFIG.A_WIDTH  32 [get_ips c_addsub_v11_0_0]
set_property CONFIG.B_WIDTH 32 [get_ips c_addsub_v11_0_0]
set_property CONFIG.ADD_MODE Add_Subtract [get_ips c_addsub_v11_0_0]
set_property CONFIG.C_IN true [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_ip_run

Creates a run for the given IP.

Syntax

`create_ip_run` **`[-force]`** **`[-quiet]`** **`[-verbose]`** *`objects`*

Returns

Nothing

Usage

Name	Description
[**-force**]	Force regeneration of products of the given IP.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	All of the IP objects (from get_ips or get_files) for which a run needs to be generated for.

Categories

Project, IPFlow

Description

Defines a synthesis and implementation run for a single IP object as specified by the **get_ips** command, or for the specified IP core file (XCI) as specified by the **get_files** command.

The IP run is used to generate the synthesis Design Checkpoint file (DCP) to support the out-of-context (OOC) IP flow, or to synthesize and implement an IP module in the OOC hierarchical design flow.

Two runs are created, one for synthesis and one for implementation. The runs are named after the IP core and the run type: *ip_name*_synth_1 and *ip_name*_impl_1.

The IP source files required to synthesize the run are copied into the IP run directory. The attributes of the run can be configured with the use of the **set_property** command.

This command returns the name of the synthesis run created for the IP module.

Arguments

-force - (Optional) Force the creation of an IP run, even if the generated output products for the specified IP are all current.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - (Required) An IP object as returned by the **get_ips** command, or an IP file (XCI) as specified by the **get_files** command.

NOTE: Only a single IP may be specified

Examples

The following example creates synthesis and implementation runs for the specified IP module:

```
create_ip_run [get_ips add1]
```

See Also

- current_run
- delete_ip_run
- launch_runs
- set_property

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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
nets	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 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 the elaborated 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 results in a one-bit bus with index value specified by the **-from** or **-to** argument

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_peripheral

Create a peripheral with a VLNV.

Syntax

```
create_peripheral [-dir arg] [-quiet]
[-verbose] vendor library name version
```

Returns

Nothing

Usage

Name	Description
[**-dir**]	Directory path for remote Peripheral to be created and managed outside the project
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
vendor	Vendor, for example xilinx.com
library	Library, for example ip
name	Name, for example myip
version	Version, for example 1.4

Categories

Project, IPFlow, CreatePeripheral

Description

Create an AXI peripheral to add to the IP repository with the specified VLNV attribute.

The AXI peripheral that is created is just a framework until interfaces have been added to the peripheral using the **add_peripheral_interface** command, and the peripheral has been generated using the **generate_peripheral** command.

Arguments

-dir *arg* - (Optional) Specify an output directory to store the AXI peripheral data files. By default, the peripheral is created and added into the source directory structure, `../project_name.srcs/sources_1/ip`, of the current project.

NOTE: If the AXI peripheral is stored outside of the current project, the specified directory should be added to the IP_REPO_PATH property of the current fileset using the **set_property** command to make the peripheral available through the IP catalog:

```
set_property IP_REPO_PATHS  {C:/Data/axi_peripheral/ourIP_2.1} [current_fileset]
update_ip_catalog -rebuild
```

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

vendor - Defines the vendor portion of the VLNV attribute that defines the location of the AXI peripheral in the IP catalog. The VLNV is the *Vendor:Library:Name:Version* string which identifies the IP in the catalog.

library - The library portion of the VLNV attribute.

name - The name portion of the VLNV attribute.

version - The version portion of the VLNV attribute.

Example

The following example creates a new AXI peripheral, with the VLNV attribute as specified:

```
create_peripheral {myCompany.com} {user} {testAXI1} {1.3} -dir {C:/Data/new_periph}
```

This example creates a new AXI peripheral, with the VLNV attribute as specified, and captures the peripheral object in a Tcl variable for later processing, then adds AXI slave interfaces to the peripheral:

```
set perifObj [ create_peripheral {myCompany.com} {user} {testAXI1} {1.3} \
-dir {C:/Data/new_periph} ]
add_peripheral_interface {S0_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
add_peripheral_interface {S1_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
add_peripheral_interface {S2_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
```

See Also

- add_peripheral_interface
- generate_peripheral
- write_peripheral

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
[**-from**]	Starting bus index
[**-to**]	Ending bus index
-direction	Pin direction Values: IN, OUT, INOUT
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
pins	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 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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	Name of the port
[*negative_name*]	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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]`** **`[-in_memory]`** **`[-ip]`** **`[-quiet]`** **`[-verbose]`** `[`*`name`*`]` `[`*`dir`*`]`

Returns

New project object

Usage

Name	Description
[**-part**]	Target part
[**-force**]	Overwrite existing project directory
[**-in_memory**]	Create an in-memory project
[**-ip**]	Default GUI behavior is for a managed IP project
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*name*]	Project name
[*dir*]	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**.

-in_memory - (Optional) Specifies that the project should be created in the in-memory database of the Vivado Design Suite only. This project will not result in a project file or directory structure being written to disk. The purpose of the in-memory project is to allow properties usually associated with a project to be associated with the in-memory design. For more information, refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)*.

-ip - (Optional) Create a project for the Managed IP flow for exploring IP in the IP catalog, customizing IP, and managing a repository of configured IP. Refer to the *Vivado Design Suite User Guide: Designing with IP (UG896)* for more information on the Managed IP flow.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 project for the Manage IP flow in the specified directory:

```
create_project -ip manageIP C:/Data
```

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 [-description arg] [-type arg] [-enum_values args]
[-default_value arg] [-file_types args] [-display_text arg] [-quiet]
[-verbose] name class
```

Returns

The property that was created if success, "" if failure

Usage

Name	Description
[**-description**]	Description of property
[**-type**]	Type of property to create; valid values are: string, int, long, double, bool, enum, file Default: string
[**-enum_values**]	Enumeration values
[**-default_value**]	Default value of type string
[**-file_types**]	File type extensions (without the dot)
[**-display_text**]	Text to display when selecting the file in file browser
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Name of property to create
class	Object type to create property for; valid values are: design, net, cell, pin, port, pblock, interface, fileset

Categories

PropertyAndParameter, XDC

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

-description *arg* - (Optional) Provide a description of the property being created. The description will be returned by the Vivado Design Suite help system when the property is queried.

-type *arg* - (Optional) The type of property to create. There allowed property types include:

- **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 short four-byte signed integer values with a range of -2,147,483,648 to 2,147,483,647. If a floating point value is specified for an **int** property type, the Vivado tool will return an error.
- **long** - Specifies signed 64-bit integers with value range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. If a floating point value is specified for an **long** 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 yes) or false (0, or no) value.
- **enum** - An enumerated data type, with the valid enumerated values specified by the **-enum_values** option.
- **string_list** - A Tcl list of string values.
- **int_list** - A Tcl list of integer values.
- **double_list** - A Tcl list of floating point values.

-enum_values *args* - (Optional) A list of enumerated values that the property can have. The list must be enclosed in braces, {}, with values separated by spaces. This option can only be used with **-type enum**.

-default_value *args* - (Optional) The default value to assign to the property. This option can be used for **string**, **int**, **bool**, and **enum** type properties.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, Pblock, interface, and fileset.

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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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: Valid Range (1-100) 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, Timing

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 between 1 and 100, 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 fileset.

Syntax

`create_sysgen` **`[-quiet]`** **`[-verbose]`** *name*

Returns

Name for the new sub module

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	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 your system design. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*name*]	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

Description

Create a new wave configuration object in the current simulation, and open the waveform configuration in the Vivado IDE. This will make the new wave configuration object the current wave configuration.

In the Vivado® simulator GUI, you can work with a waveform to analyze your design and debug your code. The Wave Config file contains the list of wave objects (signals, dividers, groups, virtual buses) to display, and their display properties, plus markers. A wave configuration displays with top-level HDL objects, and can be further populated using commands like **add_wave** and **add_wave_divider**. Any changes made to a wave configuration can be saved to a Wave Config file with the **save_wave_config** command.

This command returns the name of the waveform configuration created, or an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) The name to give to the wave configuration object. If no *name* is specified, an untitled waveform configuration will be created, called "Untitled #", where # represents a numerical sequence beginning at 1.

Examples

The following example creates a new wave configuration object with the specified name:

```
create_wave_config testbench1
```

See Also

- close_wave_config
- current_wave_config
- get_wave_configs
- open_wave_database
- open_wave_config
- save_wave_config

create_xps

Create embedded source for XPS and add to the source fileset (Not supported anymore. Please use Vivado IP integrator.).

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
name	Source name

Categories

Project

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*design*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_bd_design design_1
```

See Also

- current_bd_instance
- 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
[*instance*]	Name of current cell instance to be set

Categories

IPIntegrator

Description

Set or get the current hierarchical cell instance in the current IP Integrator subsystem design, as defined by **current_bd_design**. The current instance is referenced from the top-level of the subsystem design hierarchy, or "/".

This command returns the current IP Integrator cell instance object, or returns nothing if the command failed.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 IP Integrator hierarchical cell to be set as the current instance of the subsystem 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 subsystem 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 in the subsystem design to the specified module:

```
current_bd_instance module2
```

The following example returns the current instance:

```
current_bd_instance .
```

This example resets the current instance of the subsystem design to the top level of the hierarchy:

```
current_instance
/
```

See Also

current_bd_design

current_board

Get the current board object.

Syntax

```
current_board [-quiet] [-verbose]
```

Returns

Current board object

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_board_part

Get the current board_part object.

Syntax

```
current_board_part [-quiet] [-verbose]
```

Returns

Current board_part object

Usage

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

Categories

Object, Project, Board

Description

Return the board part used in the current project or design.

The board part provides a representation of the Xilinx FPGA device in the context of the board-level system, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces. The board part file stores information regarding board attributes. The file, called `board_part.xml`, is located in the `data/boards/board_parts` folder of the Vivado Design Suite installation area.

The board part is defined as part of a Targeted Reference Design (TRD), when the project is defined; or by setting the BOARD_PART 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 part with the **set_property** command, the target part is also changed to match the part required by the specified BOARD_PART property.*

The **current_board_part** command returns the NAME property of the current board part. The command returns a warning when the project targets a Xilinx FPGA device instead of a TRD and board, or when the BOARD_PART property has not been defined.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_PART property for the current project, then reports the board part in use by the project:

```
set_property BOARD_PART xilinx.com:kc705:part0:1.0 [current_project]
current_board_part
   xilinx.com:kc705:part0:1.0
```

This example shows the results of setting the BOARD_PART property, causing the target part to be changed as well. The target part is changed automatically, and a warning is returned:

```
set_property BOARD_PART xilinx.com:ac701:part0:1.0 [current_project]
   WARNING: [Project 1-153] The current project part 'xc7k325tffg900-2' does not
   match with the 'XILINX.COM:AC701:PART0:1.0' board part settings. The project
   part will be reset to 'XILINX.COM:AC701:PART0:1.0' 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_PART and PART property on the **current_project** to see the changes

See Also

- current_project
- get_board_part_interfaces
- get_board_part_pins
- get_board_parts
- 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
[*design*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*fileset*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Description

Set or return the current Xilinx FPGA device targeted by the Hardware Manager in the Vivado Design Suite for programming and debug.

The hardware target is a system board containing a JTAG chain of one or more Xilinx FPGA devices that you can program with a bitstream file, or use to debug your design. Connections between hardware targets on the system board and the Vivado Design Suite are managed by the **vcse_server** application. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a list of supported JTAG download cables and devices.

Each hardware target can have one or more Xilinx FPGA devices to program, or to use for debugging purposes. The current device is specified or returned by the **current_hw_device** command.

To access a Xilinx FPGA device through the Hardware Manager, you must use the following Tcl command sequence:

1. **open_hw** - Opens the Hardware Manager in the Vivado Design Suite.
2. **connect_hw_server** - Makes a connection to a local or remote Vivado CSE server application.
3. **current_hw_target** - Defines the hardware target of the connected server.
4. **open_hw_target** - Opens a connection to the hardware target.
5. **current_hw_device** - Specifies the Xilinx FPGA device to use for programming and debugging.

After connecting to the appropriate hardware device, you can program the device with a bitstream file using the **program_hw_device** command, and debug the device using any of a number of Hardware Manager Tcl commands. To interactively debug the device open the Hardware Manager in the Vivado Design Suite IDE.



IMPORTANT: *You can use the **current_hw_server**, **current_hw_target**, and **current_hw_device** commands to set the hardware for programming and debugging the design. You should exercise care when using these commands to insure that the current server, target, and device are in sync. The current device should be found on the current target, which should be found on the current server*

This command returns the current hardware device as an object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_device - (Optional) Specify the hw_device object to set as the current device for programming and debug. The hw_device must be specified as an object as returned by the **get_hw_devices** command. If the hardware device is not specified, the **current_hw_device** will be returned.

Example

The following example returns the currently targeted hw_device:

```
current_hw_device
```

This example sets the **current hw_device** object, then sets the PROGRAM.FILE property for the device, and programs the device:

```
current_hw_device [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/Data/design.bit} [current_hw_device]
program_hw_devices [current_hw_device]
```

See Also

- connect_hw_server
- current_hw_target
- get_hw_devices
- get_hw_targets
- open_hw_target

current_hw_ila

Get or set the current hardware ILA.

Syntax

`current_hw_ila` `[`**`-quiet`**`]` `[`**`-verbose`**`]` `[`*`hw_ila`*`]`

Returns

Hardware ILA

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_ila*]	hardware ILA

Categories

Hardware

Description

Set or return the current hardware ILA debug core targeted by the Hardware Manager in the Vivado Design Suite for programming and debug.

The Integrated Logic Analyzer (ILA) debug core lets you perform in-system debug of implemented designs, or design bitstreams, on a programmed Xilinx FPGA device. The ILA core includes many advanced features of modern logic analyzers, including boolean trigger equations, and edge transition triggers. You can use the ILA core to probe specific signals of the design, to trigger on programmed hardware events, and capture data from the Xilinx FPGA device in real-time. Refer to *LogiCORE IP Integrated Logic Analyzer (PG172)* for details of the ILA core.

ILA debug cores can be added to a design instantiating an ILA core from the IP catalog into the RTL design, or using the **create_debug_core** command to add the ILA core to the synthesized netlist. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on adding ILA debug cores to the design.

After generating a bitstream from the design, and programming the device with the **program_hw_devices** command, the ILA debug cores in the design are accessible from the Hardware Manager using the **get_hw_ilas** command. The debug probes assigned to the ILA debug cores in the design can be returned with the **get_hw_probes** command.

This command returns the current hardware ILA core as an object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_ila - (Optional) Specify the hw_ila object to set as the current debug core for programming and debug. The hw_ila must be specified as an object as returned by the **get_hw_ilas** command. If a ILA debug core is not specified, the **current hw_ila** will be returned.

Example

The following example returns the current hardware ILA debug core:

```
current_hw_ila
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_ilas
- get_hw_ila_datas

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

Description

Set or return the current ILA debug core data object.

The ILA data object is created in the Vivado logic analyzer using the **upload_hw_ila_data** command, or the **read_hw_ila_data** command. By default, the current hw_ila_data object is the latest one created by the Vivado logic analyzer. The **current_hw_ila_data** command can be used to change that object.

The ILA debug core captures sample data in real-time as the hardware device runs, based on the event triggers or capture conditions defined on the hw_ila object. The hw_ila object triggers on the hw_device are armed by the **run_hw_ila** command.

The ILA data object can be displayed in the waveform window of the Vivado tools logic analyzer using the **display_hw_ila_data** command. You can also write the ILA data to disk with the **write_hw_ila_data** command to save the ILA debug information for later user and analysis.

This command returns the captured hardware ILA debug core data as an object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_ila_data - (Optional) Specify the hw_ila_data object to set as the current data object for programming and debug. The hw_ila_data object must be specified as an object as returned by the **get_hw_ila_datas** command. If the hardware ILA data is not specified, the **current_hw_ila_data** object will be returned.

Example

The following example returns the current data object for the hardware ILA debug core:

```
current_hw_ila_data
```

See Also

- current_hw_ila
- display_hw_ila_data
- get_hw_ilas
- get_hw_ila_datas
- read_hw_ila_data
- run_hw_ila
- write_hw_ila_data

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

Description

Defines the current hardware server from the list of hardware servers that are connected to the Vivado Design Suite, or returns the currently connected hardware server object.

Hardware servers are instances of the Vivado tools CSE server (**vcse_server**) application running remotely, or on the local machine. The hardware server manages connections to a hardware target, such as a hardware board containing a JTAG chain of one or more Xilinx FPGA devices to be used for programming and debugging your FPGA design.

Hardware servers are connected to the Vivado Design Suite with the **connect_hw_server** command. The current hardware server, and the current hardware target and device are the focus of most Hardware Manager Tcl commands. The current target and device can be defined by the use of the **current_hw_target** and **current_hw_device** commands.

NOTE: There is usually a current hw_server defined, either the last connected hardware server, or one you have defined with this command. However, if you disconnect the current hardware server, you will need to define a new current hw_server object

You can get a list of connected hardware servers using the **get_hw_servers** command. You can get a list of available hardware targets and devices using the **get_hw_targets** and **get_hw_devices** commands respectively.

This command returns a hw_server object. If the *hw_server* is specified as part of the **current_hw_server** command, the specified server is defined as the current hardware server and that object is returned. If no server is specified, the **current_hw_server** command returns the current hardware server object.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_server - (Optional) The hw_server to set as the current hardware server. If the server is not specified, the command will return the currently defined hw_server. The hardware server can be specified by name, or specified as a hw_server object returned by the **get_hw_servers** command.

Example

The following example sets the current hardware server to the specified name:

```
current_hw_server picasso
```

The following returns the current hardware server:

```
current_hw_server
```

See Also

- connect_hw_server
- current_hw_device
- current_hw_target
- disconnect_hw_server
- get_hw_servers
- get_hw_targets
- refresh_hw_server

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

Description

After opening the Hardware Manager in the Vivado Design Suite, and connecting to the Vivado tools CSE server (**vcse_server**) using the **connect_hw_server** command, you will need to set the hardware target. This command sets or returns the current hardware target.

The hardware target is a system board containing a JTAG chain of one or more Xilinx FPGA devices that you can program with a bitstream file, or use to debug your design. Connections between hardware targets on the system board and the Vivado Design Suite are managed by the **vcse_server** application. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a list of supported JTAG download cables and devices.

The available hardware targets are defined when the **connect_hw_server** command is used to connect to a **vcse_sever**. You can return a list of the available hardware targets using the **get_hw_targets** command, and define the current hardware target using this command.

If the *hw_target* is specified as part of the **current_hw_target** command, the specified target is defined as the current hardware target and that object is returned. If no hardware target is specified, the **current_hw_target** command returns the current hardware target object.

Each hardware target can have one or more Xilinx FPGA devices to program, or to use for debugging purposes. The current device is specified or returned by the **current_hw_device** command. After specifying the current hardware target, you can open the connection through the hardware target, to the Xilinx FPGA device using the **open _hw_target** command.



IMPORTANT: *You can use the **current_hw_server**, **current_hw_target**, and **current_hw_device** commands to set the hardware for programming and debugging the design. You should exercise care when using these commands to insure that the current server, target, and device are in sync. The current device should be found on the current target, which should be found on the current server*

This command returns the current hardware target as an object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_target - (Optional) Specify the hw_target object to set as the current_hw_target for programming and debug. The hw_target must be specified as an object as returned by the **get_hw_targets** command. If the hardware target is not specified, the **current_hw_target** will be returned.

Example

The following example returns the available hardware targets on the connected hardware servers, and sets the **current hw_target** to the specified target:

```
get_hw_targets
   trumpet/xilinx_tcf/Digilent/210203327985A
   picasso/xilinx_tcf/Digilent/210203368518A
current_hw_target [lindex [get_hw_targets] 1]
```

See Also

- close_hw_target
- current_hw_device
- get_hw_devices
- get_hw_targets
- open_hw_target
- refresh_hw_target

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
[*instance*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*hdl_scope*]	Suspend message limits during command execution Default: NULL

Categories

Simulation

Description

Return the current scope in the current simulation, or set the current scope to the specified HDL scope.

The **current_scope** command returns the name of the current simulation scope.

If *hdl_scope* is supplied then, the current scope is set to the specified scope.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_scope - (Optional) Specify an HDL scope to assign as the current scope of the current simulation. The scope can be specified as an absolute hierarchical path name (such as `/tb/UUT`), a relative path name (`/uut, gt, ./gt, uut/fg`) of a scope, or an HDL scope object, returned by the **get_scopes** command.



TIP: *If you specify '/' as the HDL scope, the scope is reset to the top-level scope in the current simulation*

Examples

The following example sets the current scope to the specified HDL scope:

```
current_scope /testbench/dut
```

This example returns the current scope name to console:

```
current_scope
```

See Also

- get_scopes
- report_scopes

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Simulation

Description

Returns the current simulation time to the Tcl Console or Vivado Design Suite Tcl shell.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 time of the current simulation:

```
current_time
```

See Also

- restart
- run
- stop

current_vcd

Return the current VCD object or make VCDObject the current VCD object.

Syntax

`current_vcd` `[`**`-quiet`**`]` `[`**`-verbose`**`]` `[`*`VCDObject`*`]`

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*VCDObject*]	VCDObject Default: NULL

Categories

Simulation

Description

Defines the current Value Change Dump (VCD) object, or returns the name of the current VCD object in the current simulation.

A VCD file must be opened and assigned to a VCD object using the **open_vcd** command in order for there to be a current VCD object.

This command returns the current VCD object.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

VCDObject - (Optional) The name of the VCD object to set as the current object. If a *VCDObject* is not specified, the command returns the current VCD object of the active simulation.

Examples

The following example sets the specified VCD object as current:

```
current_vcd vcd2
```

See Also

open_vcd

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

Description

Set or get the current wave configuration object for the current simulation.

In the Vivado® simulator GUI, you can work with a waveform to analyze your design and debug your code. A wave configuration displays with top-level HDL objects, and can be further populated using commands like **add_wave** and **add_wave_divider**.

This command returns the name of the current wave configuration object.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

wcfgObj - (Optional) Specify a wave configuration object to set as the current wave configuration. The wave configuration object can either be specified by name, or as an object returned by the get_wave_configs command.

Examples

The following example gets the testbench wave config object and makes it the current wave configuration in the simulation:

```
current_wave_config [get_wave_config testbench]
```

See Also

- create_wave_config
- get_wave_configs
- open_wave_config

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
objects	The objects to be deleted

Categories

IPIntegrator

Description

Delete specified objects from the current IP Integrator subsystem design.

Objects must be passed directly to the **delete_bd_objs** command, and not simply referenced by the object name. Pins are passed to the command by **get_bd_pins**, for instance, rather than by pin name.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - A list of objects to delete from the current IP Integrator subsystem design.

Example

The following example deletes the various objects from the current subsystem design:

```
delete_bd_objs [get_bd_nets /Net] [get_bd_nets /vidout_1_vtg_ce] \
[get_bd_nets /newMod1/aclk_1] [get_bd_ports /addr] [get_bd_cells /vidOut_1]
```

The following example deletes the same objects, but uses multiple **delete_bd_objs** commands to clarify the objects that are being deleted by grouping them by type:

```
delete_bd_objs [get_bd_nets /Net] [get_bd_nets /vidout_1_vtg_ce] \
[get_bd_nets /newMod1/aclk_1]
delete_bd_objs [get_bd_ports /addr]
delete_bd_objs [get_bd_cells /vidOut_1]
```

See Also

- get_bd_addr_segs
- get_bd_addr_spaces

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
name	Name for the set of results to clear

Categories

Report

Description

Clear the results of the specified **report_clock_networks** report from the named result set.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 clock networks results to be cleared.

Examples

The following example clears the specified results set from memory:

```
delete_clock_network_results ClkNets
```

See Also

report_clock_networks

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
cores	Debug cores to delete

Categories

Debug

Description

Removes Vivado lab tools debug cores from the current project that were added by the **create_debug_core** command.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
ports	Debug ports to delete

Categories

Debug

Description

Deletes ports from Vivado lab tools 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	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 get_drc_checks to determine the correct name to use. Only user-defined DRC checks 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 returns nothing if successful, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 LJH-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 returns nothing if successful, or returns an error if it fails.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_ruledeck 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_ruledeck
```

See Also

- create_drc_ruledeck
- 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
fileset	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

-merge *arg* - (Optional) Specify a different fileset to merge the files from the deleted fileset into. If the **-merge** option is not specified, then all files in the fileset are removed from the project.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_fileset 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_fileset
- current_fileset

delete_hw_axi_txn

Delete hardware AXI Transaction objects.

Syntax

`delete_hw_axi_txn` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`hw_axi_txns`* `...`

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_axi_txns	hardware AXI Transaction object to delete

Categories

Hardware

delete_hw_bitstream

Removes the HW Bitstream object from a list of hardware devices.

Syntax

`delete_hw_bitstream` `[`**`-of_objects`** *`args`*`]` `[`**`-quiet`**`]` `[`**`-verbose`**`]`

Returns

Hardware devices

Usage

Name	Description
[**-of_objects**]	Get 'hw_bitstream' objects of these types: 'hw_device'.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Hardware, Object

Description

This command deletes the hw_bitstream object from the specified hw_device objects.

This clears the PROGRAM.HW_BITSTREAM and PROGRAM.FILE properties on the hw_device objects, and deletes the hw_bitstream object.

Arguments

-of_objects *arg* - (Optional) Delete the hardware bitstream object (hw_bitstream) associated with the specified hardware devices (hw_devices). The targets must be specified as objects using the **get_hw_devices** or the **current_hw_device** commands.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 deletes the hw_bitstream object associated with the current hardware device:

```
delete_hw_bitstream -of_objects [current_hw_device]
```

See Also

- create_hw_bitstream
- current_hw_device
- get_hw_devices
- program_hw_devices
- set_property
- write_bitstream

delete_hw_cfgmem

Removes hw_cfgmem object from memory.

Syntax

`delete_hw_cfgmem` `[`**`-quiet`**`] [`**`-verbose`**`] [`*`cfgmem`*`]`

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*cfgmem*]	Valid hw_cfgmem object

Categories

Hardware

Description

Removes the specified hw_cfgmem object from the current hw_device.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

cfgmem - (Required) Specify a hw_cfgmem object to remove from the current hw_device. The hw_cfgmem must be specified as an object, rather than simply by name.

NOTE: You can use the **get_property PROGRAM.HW_CFGMEM** command on the **current_hw_device** to specify the hw_cfgmem object, or you can save the hw_cfgmem object to a Tcl variable when you create it with the **create_hw_cfgmem** command.

Example

The following example removes the hw_cfgmem object associated with the current hw_device, using the PROGRAM.HW_CFGMEM property to identify the object:

```
delete_hw_cfgmem [get_property PROGRAM.HW_CFGMEM [current_hw_device]]
```

See Also

- create_hw_cfgmem
- current_hw_device
- get_property

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
interfaces	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_ip_run

Deletes the block fileset and run associated with a given IP.

Syntax

`delete_ip_run [-force] [-quiet] [-verbose]` *`objects`*

Returns

Nothing

Usage

Name	Description
[**-force**]	Force the deletion of the block fileset and run.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	All of the IP objects (from get_ips or get_files) for which the block fileset and run will be deleted.

Categories

Project, IPFlow

Description

Deletes the out-of-context (OOC) synthesis and implementation runs for the specified IP module.

The contents of the run directory are deleted from the project as well as the run. However, the output products created by the run and copied to the IP sources folder, the DCP file and Verilog and VHDL structural netlists, are not deleted from the project. You must use the **reset_target** or **generate_target** command to update the IP output products.

IMPORTANT: *The command requires an IP object as specified by the **get_ips** or **get_files** command, and will not delete a run based on either the name of the run, or a run object as returned by **get_runs***

Arguments

-force - Force the deletion of the run.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 object or file to delete the run from. An IP can be specified as an object using the **get_ips** command, or as a file (XCI) using the **get_files** command.

Examples

The following example deletes the OOC synthesis and implementation runs from the specified IP module:

```
delete_ip_run [get_ips add1]
```

NOTE: In this example, all run results will also be removed from the run directory on the hard drive.

See Also

- create_ip_run
- generate_target
- get_files
- get_ips
- reset_target

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
macros	Macros to delete

Categories

XDC

Description

Delete one or more macro defined by the **create_macro** command.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

macros - (Required) Specify the name or names of the macros to delete.

Examples

The following example creates a macro called :

```
delete_macros usbMacro1
```

See Also

create_macro

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
pblocks	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
rpm	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_runs

Delete existing runs.

Syntax

`delete_runs [-noclean_dir] [-quiet] [-verbose]` *runs*

Returns

Nothing

Usage

Name	Description
[**-noclean_dir**]	Do not remove all output files and directories from disk
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
runs	Run to modify

Categories

Project

Description

Deletes the specified runs 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 the synthesis or implementation runs to delete from the project.

Examples

The following example deletes the first_pass run from the project:

```
delete_runs 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_runs -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
name	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:

- clock_interaction - Delete the named **report_clock_interaction** report.
- check_timing - Delete the named **check_timing** report.
- datasheet - Delete the named report_datasheet report.
- pulse_width - Delete the named **report_pulse_width** report.
- slack_histogram - Delete the named **create_slack_histogram** report.
- timing_path - Delete the named **report_timing** 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 timing results to be cleared.

Examples

The following example clears the specified results set from memory:

```
delete_timing_results -type clock_interaction clkNets
```

See Also

- check_timing
- create_slack_histogram
- report_clock_interaction
- report_datasheet
- 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 SSO1
```

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
hdl_object	The hdl_object or hdl_scope to describe

Categories

Simulation

Description

Describe an HDL object (variable, signal, wire, or reg) by printing type and declaration information, as well as path, and file information for the HDL source of the specified objects.



TIP: *The **describe** command works for a single HDL object. Use the **report_objects** command for a brief report on multiple 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 command returns the description of specified HDL objects, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 describe.

NOTE: Objects can be specified by name, or returned as objects by the **get_objects** command

Examples

The following example shows how the objects description depends on the scope of the current simulation:

```
current_scope testbench
/testbench
describe leds_n
    Signal: {leds_n[3:0]}
    Path: {/testbench/leds_n}
    Location: {File "C:/Data/ug937/sim/testbench.v" Line 9}
current_scope dut
/testbench/dut
describe leds_n
    Port(OUT): {LEDS_n[3:0]}
    Path: {/testbench/dut/LEDS_n}
    Location: {File "C:/Data/sources/sinegen_demo.vhd" Line 42}
```

See Also

- current_scope
- get_objects
- report_objects

detect_hw_sio_links

Automatically detect links between RX and TX endpoints. Create a new link group to contain the links.

Syntax

```
detect_hw_sio_links [-force] [-quiet] [-verbose]
```

Returns

A new hardware SIO link group of found links

Usage

Name	Description
[**-force**]	Remove all existing links before detecting links
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Hardware

Description

Automatically detects existing or previously defined communication pathways between GT transmitters and receivers that are defined on the open hardware target.

You can use this command if you change board connections while the serial I/O analyzer is running. The detection algorithm uses changing transmit patterns and detects links on received patterns to determine how GTs are connected to one another on the open hardware target.

A transmitter or receiver of an individual GT on the IBERT debug core can only be used in one hw_sio_link at a time, so the command will not check GTs that are used in existing links. The **-force** option lets you clear all existing links before scanning the open hardware target to check all GTs.

The **detect_hw_sio_links** command defines the found links, and creates a link group to associate the new links.

This command returns the number of links found and the created hw_sio_linkgroup object, or returns an error if it fails.

Arguments

-force - (Optional) Delete existing link objects and re-detect links by examining all GTs to ensure the current hardware configuration is properly reflected in the link definitions.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 examines the IBERT debug core on the open hardware target to look for existing communication links:

```
detect_hw_sio_links
```

NOTE: Without the **-force** option, GTs used in existing links will not be examined.

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups

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
intf_net	The IntfNet that the objects connect to
objects	The objects to disconnect from the intf_net

Categories

IPIntegrator

Description

Disconnect a single interface net in the IP Integrator subsystem design from the specified objects. An interface is a grouping of signals that share a common function in the IP Integrator subsystem design.

This command lets you disconnect the specified interface net from pins or ports in the IP subsystem design, without deleting the whole net. To delete the whole net, you should use the **delete_bd_objs** command.

This command returns 0 if it is successful, and returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

intf_net - (Required) Specifies a single interface net in the IP subsystem design to disconnect from the specified objects. The net can be specified by name, or as a single object returned by the **get_bd_intf_nets** command.

objects - (Required) The list of interface pin or port objects to disconnect the net from. The interface pins and ports must be specified as design objects returned by the **get_bd_intf_pins** or **get_bd_intf_ports** commands. They cannot simply be referenced by name.

Example

The following example disconnects the only interface net in the IP subsystem design, from all interface pin and port objects that are connected to it:

```
disconnect_bd_intf_net [get_bd_intf_nets] \
[get_bd_intf_pins -of_objects [get_bd_intf_nets]] \
[get_bd_intf_ports -of_objects [get_bd_intf_nets]]
```

NOTE: In this example you know there is only one interface net in the design, or an error would be returned by the **disconnect_bd_intf_net** command, which only accepts a single interface net

See Also

- connect_bd_intf_net
- connect_bd_net
- delete_bd_objs
- disconnect_bd_intf_net
- get_bd_nets
- get_bd_pins
- get_bd_ports

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
net	The Net that the objects connect to
objects	The objects to disconnect from the net

Categories

IPIntegrator

Description

Disconnect a single net in the IP Integrator subsystem design from the specified objects.

This command lets you disconnect the specified nets from pins or ports in the IP subsystem design, without deleting the whole net. To delete the whole net, you should use the **delete_bd_objs** command.

This command returns 0 if it is successful, and returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - (Required) Specifies a single net in the IP subsystem design to disconnect from the specified objects. The net can be specified by name, or as a single object returned by the **get_bd_nets** command.

objects - (Required) The list of pin or port objects to disconnect the net from. The pins and ports must be specified as design objects returned by the **get_bd_pins** or **get_bd_ports** commands. They cannot simply be referenced by name.

Example

The following example disconnects the net, here specified by name, from the specified pin objects:

```
disconnect_bd_net /vidout1_locked [get_bd_pins {vidOut1/locked newMod1/t1}]
```

See Also

- connect_bd_intf_net
- connect_bd_net
- delete_bd_objs
- disconnect_bd_intf_net
- get_bd_nets
- get_bd_pins
- get_bd_ports

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
[**-channel_index**]	Disconnect the net at channel index
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
port	Debug port name

Categories

Debug

Description

Disconnect signals from the debug ports.

Signals from the Netlist Design are connected to ports of a ILA 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 PROBE1 port of myCore:

```
disconnect_debug_port -channel_index 2 myCore/PROBE1
```

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/PROBE1
```

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

Description

Disconnect the current or specified Vivado tools CSE server from the Vivado Design Suite.

The current hardware server is either the last connected hardware server, or one you have manually defined with the **current_hw_server** command. If you disconnect the current hardware server, there will be no defined current hardware server until you define a new current hw_server object.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_server - (Optional) The hw_server to disconnect from the Vivado Design Suite. If the server is not specified, the **current_hw_server** will be disconnected. The hardware server can be specified by name, or specified as a hw_server object returned by the **get_hw_servers** or **current_hw_server** commands.

Example

The following example disconnects the specified hardware server from the Vivado Design Suite:

```
disconnect_hw_server picasso
```

See Also

- connect_hw_server
- current_hw_server
- get_hw_servers
- refresh_hw_server

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 disconnect, remove the net and any pin/net chain up to the pin on any primitive instance as long as each object in the chain has only 1 remaining connection.
-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 the elaborated RTL design.

Arguments

-prune - (Optional) Remove hierarchical pins, ports, or nets that are left unconnected after disconnecting the specified nets. This will not remove the pins specified by **-objects**, which are disconnected, but removes the net and pins or ports connected to the specified pins.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Using the following connection network:

```
leaf_cell1/pin1 > net1 > block1/pin1 > topnet < block2/pin1 < net2 < leaf_cell2/pin1
```

This example first disconnects the signal, topnet, from block1/pin1, and then prunes topnet, block2/pin1, and net2, but does not prune leaf_cell2/pin1:

```
disconnect_net -prune -net topnet -objects [get_pins block1/pin1]
```

NOTE: net2 is not removed, because block1/pin1 is not pruned as part of the **-prune** option

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**]	Import alternate wave config file
[**-reset**]	Force reset wave config file to default configuration
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_ila_data*]	List of hardware ILA data objects. Default: Current hardware ILA data

Categories

Hardware

Description

This command is intended for use with the graphical user interface of the Vivado Design Suite logic analyzer feature. It displays the specified ILA debug core data object in a wave config window of the Vivado logic analyzer.

The ILA debug sample data is acquired from a running device using the **upload_hw_ila_data** command. This creates a hw_ila_data object that can be written to a file on disk using the **write_hw_ila_data** command. This command reads that ILA data file.

The display characteristics of the ILA debug core in the waveform window are determined by the Wave Config file. The Wave Config file contains just the list of wave objects (signals, dividers, groups, virtual buses) to display, and their display properties, plus markers.

A wave configuration object is created in the Vivado logic analyzer with the **create_wave_config** command. A Wave Config file is written to disk by the use of the **save_wave_config** command, and can be opened with the **open_wave_config** command.

The **open_wave_config** command opens a Wave Config file and maps it to the data source in the current simulation.

Arguments

-wcfg *arg* - (Optional) View the ILA data using the specified Wave Config file, created using the **save_wave_config** command. If the **-wcfg** option is not specified, the ILA data will display in a default wave configuration as determined by the Vivado logic analyzer.

-reset - (Optional) Reset the waveform window to the default configuration and display the specified ILA data.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_ila_data - (Optional) Specify one or more hw_ila_data objects to display. The hw_ila_data must be specified as an object as returned by the **get_hw_ila_datas** or **current_hw_ila_data** commands. If the hardware ILA data is not specified, the **current_hw_ila_data object** will be displayed.

Example

The following example reads a hw_ila_data file, and displays the resulting hw_ila_data object in the waveform window of the Vivado logic analyzer:

```
read_hw_ila_data C:/hw_ila_file.ila
display_hw_ila_data [get_hw_ila_datas hw_ila_file]
```

See Also

- current_hw_ila
- current_hw_ila_data
- upload_hw_ila_data
- get_hw_ilas
- get_hw_ila_datas
- read_hw_ila_data
- run_hw_ila
- save_wave_config

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
hw_sio_scans	hardware SIO scans

Categories

Hardware

Description

This command is intended for use with the graphical user interface of the Vivado Design Suite serial I/O analyzer feature. It displays the specified SIO scan data object, or objects, in a Scan Plots window of the Vivado IDE.

The SIO scan data can be read from a file on disk using the **read_hw_sio_scan** command, or from a hw_sio_scan object created by the **run_hw_sio_scan** command. The type of plot displayed is determined by the *scan_type* of the hw_sio_scan object.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_scans - (Optional) Specify one or more existing hw_sio_scan objects to display in Scan Plot windows in the Hardware Manager feature of the Vivado IDE.

Example

The following example reads an SIO scan data file into memory, and displays the hw_sio_scan object that is created:

```
display_hw_sio_scan [read_hw_sio_scan C:/Data/loopback1.csv]
```

See Also

- create_hw_sio_scan
- current_hw_device
- get_hw_sio_scans
- remove_hw_sio_scan
- run_hw_sio_scan
- read_hw_sio_scan
- stop_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

encrypt

Encrypt files in place with a language specific key file in IEEE 1735. no default.

Syntax

`encrypt [-quiet] [-verbose]` *`key files...`*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
key	key file to be used to encrypt
files	Files to be encrypted in place

Categories

FileIO

endgroup

End a set of commands that can be undone/redone as a group.

Syntax

```
endgroup [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_simulation

(User-written application).

Syntax

```
export_simulation [-of_objects arg] [-lib_map_path arg]
[-script_name arg] [-absolute_path] [-32bit] [-force] -directory arg
-simulator arg [-quiet] [-verbose]
```

Returns

None

Usage

Name	Description
[**-of_objects**]	Export simulation script for the specified object Default: None
[**-lib_map_path**]	Precompiled simulation library directory path. If not specified, then please follow the instructions in the generated script header to manually provide the simulation library mapping information. Default: Empty
[**-script_name**]	Output shell script filename. If not specified, then file with a default name will be created with the '.sh' extension. Default: top_module.sh
[**-absolute_path**]	Make all file paths absolute wrt the reference directory Default: 0
[**-32bit**]	Perform 32bit compilation Default: 0
[**-force**]	Overwrite previous files Default: 0
-directory	Directory where the simulation script will be exported
-simulator	Simulator for which the simulation script will be created (*name*: ies|vcs_mx)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

xilinxtclstore, projutils, user-written

Description

Export a simulation script file for the target simulator. Currently the Cadence Incisive Enterprise Simulator (**ies**) and the Synopsys VCS MX simulator (**vcs_mx**) are supported. The generated script will contain simulator commands for compiling, elaborating and simulating the design.

The command will retrieve the simulation compile order of specified objects, and export this information in a text file with the compiler commands and default options for the target simulator. The specified object can be either a simulation fileset or an IP. If the object is not specified, then the **export_simulation** command will generate the script for the simulation top.

Any verilog include directories or file paths for the files containing verilog define statements will be added to the compiler command line.

By default, the design source file and include directory paths in the compiler command line will be set relative to the "reference_dir" variable that is defined in the generated script. To make these paths absolute, specify the **-absolute_path** option.

The command will also copy data files (if any) from the fileset, or from an IP, to the output directory. If the design contains "Verilog" sources, then the generated script will also copy "glbl.v" from the software installation path to the output directory.

A default ".do" file will be created in the output directory for the target simulator that will be referred in the compiler commands in the script.

NOTE: In order to perform simulation with the generated script, the simulation libraries must be compiled first using the **compile_simlib** command, with the compiled library directory path specified, when generating this script. The generated simulation script will automatically include the setup files for the target simulator from the compiled library directory

This command returns nothing.

Arguments

-of_objects *arg* - (Optional) Specify the target object for which the simulation script file needs to be generated. The target object can be either a simulation fileset (simset) or an IP core. If this option is not specified then the command will generate simulation scripts for the current simulation fileset.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

-lib_map_path *arg* - (Optional) Specify the pre-compiled simulation library directory path where the Xilinx simulation libraries are compiled. Please see the header section in the generated script for more information.

-script_name *arg* - (Optional) Specify the name of the shell script. If this option is not specified then the filename will be generated based on the object type selected with **-of_objects** switch, with the following form:

- *simulation_top_name*_sim_*simulator*.sh
- *ip_name*_sim_*simulator*.sh

-absolute_path - (Optional) Specify this option to make source and include directory paths used in the script absolute. By default, all paths are written as relative to the directory path that is specified with the **-dir** option. A "reference_dir" variable will be set in the simulation script to the directory path that is specified with the **-directory** option.

-32bit - (Optional) Specify this option to perform 32-bit simulation. If this option is not specified then by default a 64-bit option will be added to the simulation command line.

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

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

-dir *arg* - (Required) Specify the directory path where the simulation files will be exported.

-simulator [ies | vcs_mx] - (Required) Specify the target simulator name for the simulation script. The valid simulators are **ies** and **vcs_mx**.

Examples

The following command generates a simulation script file in the current directory for the "IES" simulator:

```
export_simulation -simulator ies -directory .
```

The following command overwrites an existing script file in the current directory:

```
export_simulation -force -simulator ies -directory .
```

The following command generates a simulation script file named "test_ies.sh" in the "./test_sim" directory:

```
export_simulation -simulator ies -directory ./test_sim -script_name test_ies.sh
```

The following command generates a script file named "top_tb_sim_ies.sh" in the "./test_sim" directory for a project with simulation top set to "top_tb". The command will also copy any data files (.mif, .coe, .dat, etc) to the ./test_sim directory:

```
export_simulation -simulator ies -directory ./test_sim
```

The following command generates a script file "accum_0_sim_ies.sh" for the "accum_0" IP in the specified output directory for the "IES" simulator:

```
export_simulation -of_objects [get_files accum_0.xci] -simulator ies \
-directory test_sim
```

The following command generates a script file "accum_0_sim_vcs_mx.sh" for the "accum_0" IP in the specified output directory for the "VCS_MX" simulator:

```
export_simulation -of_objects [get_ips accum_0] -simulator vcs_mx \
-directory test_sim
```

The following command generates a script file "fifo_tb_sim_vcs_mx.sh" for the simulation fileset "sim_fifo_test" whose top is set to "fifo_tb" in the specified output directory for the "IES" simulator:

```
export_simulation -of_objects [get_filesets sim_fifo_test] -simulator ies -directory test_sim
```

The following command exports a script file "top_tb_sim_vcs_mx.sh" for the "VCS_MX" simulator in the specified output directory with the design source files compiled for 32 bit version of the simulator compiler (no 64 bit option will be added to the command line):

```
export_simulation -force -32bit -simulator vcs_mx -directory test_bft_vcs_mx
```

The following example will include "/sim_libs/ius/lin64/lib/cds.lib" file path in the "./test_sim/cds.lib" file ("INCLUDE /sim_libs/ius/lin64/lib/cds.lib") for referencing the compiled libraries for "IES" simulator:

```
export_simulation -lib_map_path "/sim_libs/ius/lin64/lib" -simulator ies \
-directory "test_sim"
```

The following example will include "/sim_libs/vcs/lin64/lib/synopsys_sim.setup" file path in the "./test_sim/synopsys_sim.setup" file ("OTHERS=/sim_libs/vcs/lin64/lib/synopsys_sim.setup") for refrencing the compiled libraries for the "VCS_MX" simulator:

```
export_simulation -lib_map_path "/sim_libs/vcs/lin64/lib" -simulator vcs_mx \
-directory "test_sim"
```

The following example generates a script file in "./test_sim/ies" directory and then compiles, elaborates and simulates the design in "IES" simulator:

```
export_simulation -lib_map_path "/sim_libs/ies/lin64/lib" -simulator ies \
-directory "./test_sim/ies"
cd test_sim/ies
./top_tb_sim_ies.sh
```

The following example generates a script file in "./test_sim/vcs_mx" directory and then compile, elaborate and simulate the design in "VCS_MX" simulator:

```
export_simulation -lib_map_path "/sim_libs/vcs/lin64/lib" -simulator vcs_mx \
-directory "./test_sim/vcs_mx"
cd test_sim/vcs_mx
./top_tb_sim_vcs_mx.sh
```

See Also

- compile_simlib
- current_fileset
- get_files
- get_ips

filter

Filter a list, resulting in new list.

Syntax

`filter` **`[-regexp]`** **`[-nocase]`** **`[-quiet]`** **`[-verbose]`** *`[objects]`* *`[filter]`*

Returns

New list

Usage

Name	Description
[**-regexp**]	Operators =~ and !~ use regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	List of objects to filter
[*filter*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. For example, in the case of the "part" object, "DEVICE", "FAMILY" and "SPEED" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 [-boundary_type 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 hierarchy boundaries. 'connected_to_hierip' functions like connected_to except that it will stop on boundaries of hierarchy that are linked to IPs.
[**-boundary_type**]	Used when source object is on a hierarchical block's pin or intf_pin. Valid values are 'upper', 'lower' or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for relation: connected_to Default: upper
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	One or more source object to start finding from

Categories

IPIntegrator

Description

Find a list of pins, ports or interfaces with a given relationship to the specified objects.

This command returns a list of pins, ports or interface objects if successful, or returns an error if it fails.

Arguments

-relation *arg* - (Required) Specifies the relationship of the objects to find with the objects specified to seed the search. Valid values are:

- **connected_to** - Find corresponding pins, ports or interfaces that are connected to the given source objects, across hierarchy boundaries.
- **connected_to_hierip** - Functions like **connected_to** except that it will stop on the hierarchical boundaries of IP cells.
- **addressable_slave** - Find pins, ports, or interfaces, that can be addressed as slaves by the specified search objects.
- **addressing_master** - Find pins, ports, or interfaces, that address the specified search objects as masters.

-boundary_type [upper | lower | both] - (Optional) Used when the specified source objects include a pin or interface pin on a hierarchical module. The default behavior is "upper", which searches the same level as the hierarchical module for related objects. If "lower", the search occurs inside the hierarchical module. If "both", then both the level of the hierarchical module, and inside the hierarchical module are searched for related objects.

NOTE: This option can only be used with **-relation connected_to**.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 use to find related objects.

Example

The following example gets pins, ports, and interfaces connected to the specified object:

```
find_bd_objs -relation connected_to \
[get_bd_pins /proc_sys_reset_1/peripheral_aresetn]
```

See Also

- create_bd_cell
- create_bd_intf_net
- create_bd_intf_pin
- create_bd_intf_port
- create_bd_net
- create_bd_pin
- create_bd_port

find_routing_path

Find a routing path between two nodes.

Syntax

`find_routing_path` **`[-allow_overlap]`** **`[-max_nodes`** *`arg`*`]` **`[-min_nodes`** *`arg`*`]`
`[-from` *`args`*`]` **`[-to`** *`args`*`]` **`[-quiet]`** **`[-verbose]`**

Returns

Nothing

Usage

Name	Description
[**-allow_overlap**]	Solution may include nodes used in existing routes.
[**-max_nodes**]	Specifies the maximum number of nodes (including from and to nodes) allowed in solution. Default: 100
[**-min_nodes**]	Specifies the minimum number of nodes (including from and to nodes) allowed in solution. Default: 2
[**-from**]	-from Start of routing path.
[**-to**]	-to End of routing path.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Object

Description

Finds a routing solution between two nodes on an unrouted, or partially routed net, in an implemented design.

This command can be used to define a routing path to assign to the FIXED_ROUTE property of a net, which can be saved to the XDC file for later reuse. Refer to *Vivado Design Suite Tutorial: Implementation (UG986)* for an example of manual routing and the use of the FIXED_ROUTE property.

You must define nodes for the start and end points of the routing path, and can specify the maximum and minimum number of nodes to use for the route path, including the start and end points. The nodes must be specified as objects returned by the get_nodes command. For unrouted net objects, since nodes have not been assigned to the net, the nodes can be found by association of the net to the bel_pin or site_pin:

- Net > Bel_Pin > Bel > Tile > Node
- Net > Site_Pin > Tile > Node

For partially routed nets, the nodes can be found associated directly to the net. Refer to the *Vivado Design Suite Properties Reference Guide (UG912)* for more information on the relationship between these objects.

The **find_routing_path** command returns a list of nodes representing the route path found from the start point to the end point, or returns "no path found" if the command runs but has no result, or returns an error if the command fails to run.

The returned list of nodes can be assigned to the FIXED_ROUTE property using the set_property command as shown in the example.



TIP: *The **report_property** command does not return the string of the FIXED_ROUTE property. Use the **get_property** command instead*

Arguments

-allow_overlap - (Optional) Enable a loose style of routing which can create conflicts that must be later resolved. These overlapping routes will need to be cleaned up to eliminate routing conflicts. Route conflicts can be identified using the **report_route_status** command.

-max_nodes *arg* - (Optional) Indicates the maximum number of nodes the route can contain, including the **-from** node and the **-to** node. The default is 100.

-min_nodes *arg* - (Optional) Indicates the minimum number of nodes the route can contain, including the **-from** node and the **-to** node. The default is 2, and the value specified must be >= 2.



TIP: *This option can be used to generate a meandering route that will provide some added timing delay.*

-from *arg* - (Optional) The starting node of the route path. Nodes must be specified as objects returned by the get_nodes command.

-to *arg* - (Optional) The ending node of the route path. Nodes must be specified as objects returned by the get_nodes command.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 finds a routing path for the specified net, using one end as the -from point and the other end as the -to point, and assigns that path to the specified Tcl variable. Then it uses that Tcl variable to assign the path to the FIXED_ROUTE property of the net:

```
set fndPath [find_routing_path -from [lindex [get_nodes -of \
[get_site_pins -of [get_nets wbOutputData_OBUF[14]]]] 0] -to \
[lindex [get_nodes -of  [get_site_pins -of [get_nets wbOutputData_OBUF[14]]]] 1]]
set_property FIXED_ROUTE $fndPath [get_nets wbOutputData_OBUF[14]]
```

See Also

- get_bel_pins
- get_nets
- get_nodes
- get_property
- get_site_pins
- report_property
- report_route_status
- set_property

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`*`]` `[`**`-return_file_paths`**`]` `[`**`-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
[**-return_file_paths**]	For each top returned, also include the associated file path. The returned value will be a single list of strings, alternating top names and file paths.
[**-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 **lindex** 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 =~ *usbf*} ] ]
usbf_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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_mem_files

Write all the simulation .mem files.

Syntax

`generate_mem_files [-force] [-quiet] [-verbose]` *`directory`*

Returns

The name of the directory

Usage

Name	Description
[**-force**]	Overwrite existing .mem files
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
directory	Directory for exporting .mem files. Values: A directory with alphanumeric characters.

Categories

FileIO, Simulation

generate_peripheral

Generate output products for peripheral object.

Syntax

```
generate_peripheral [-driver] [-example_design] [-bfm_example_design]
[-debug_hw_example_design] [-enable_interrupt] [-force] [-quiet]
[-verbose] peripheral
```

Returns

Nothing

Usage

Name	Description
[**-driver**]	Generate driver for peripheral.
[**-example_design**]	Generate all supported example designs for peripheral.
[**-bfm_example_design**]	Generate bfm simulation example design for peripheral.
[**-debug_hw_example_design**]	Generate debug hardware example design for peripheral.
[**-enable_interrupt**]	Generate peripheral with interrupt suppport.
[**-force**]	Overwrite the existing IP in the repository.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
peripheral	peripheral object

Categories

Project, IPFlow, CreatePeripheral

Description

Generate the output products for the specified peripheral object. The output products are written to the IP repository location specified when the IP is created by the **create_peripheral** command, under the name of the IP as specified at creation.

Arguments

-driver - (Optional) Create software driver files containing offsets of software addressable registers in the generated peripheral, as well as masks and register access macros or utility functions. The software driver self test example file contains self test example code to test various hardware features of the peripheral.

-example_design - (Optional) Write example design data for the specified peripheral. This includes the `design.tcl` to create a block design incorporating the new peripheral in IP Integrator, and a test bench, called `design_tb.v`, for simulating the example design.

NOTE: This option is not currently supported for AXI peripherals that implement the AXI stream interface as defined by the **-axi_type** option of the **add_peripheral_interface** command.

-bfm_example_design - (Optional) Create a TCL script to generate a block design using the IP integrator feature of the Vivado Design Suite ad a bus functional model (BFM) test bench to test the read and write operations of the AXI peripheral.

NOTE: AXI4 BFM requires a license for use during simulation

-debug_hw_example_design - (Optional) Create a Tcl script to generate a block design for debugging the AXI peripheral using the JTAG-to-AXI debug core in the Hardware Manager feature of the tool. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on working with the Hardware Manager.

-enable_interrupt - (Optional) Add an interrupt pin and supporting logic, to enable the interrupt operation on the peripheral.

-force - (Optional) Overwrite any existing output products in the IP repository even if they are current.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

peripheral - (Required) The peripheral object to generate output products for. The peripheral is created with the **create_peripheral** command, and should be captured in a Tcl variable at the time it is created to facilitate further processing by this and other related commands. See the example below.

Example

This example creates a new AXI peripheral, with the VLNV attribute as specified, and captures the peripheral object in a Tcl variable for later processing, then adds AXI slave interfaces to the peripheral, and generates the output products for the peripheral:

```
set perifObj [ create_peripheral {myCompany.com} {user} {testAXI1} {1.3} \
-dir {C:/Data/new_periph} ]
add_peripheral_interface {S0_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
add_peripheral_interface {S1_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
generate_peripheral -driver -bfm_example_design -enable_interrupt $perifObj
write_peripheral $perifObj
set_property ip_repo_paths  C:/Data/new_periph [current_fileset]
update_ip_catalog -rebuild
```

See Also

- add_peripheral_interface
- create_peripheral
- write_peripheral

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
name	List of targets to be generated, or 'all' to generate all supported targets
objects	The objects for which data needs to be generated

Categories

Project, 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 block designs (`.bd`). 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 (BD) imported from IP Integrator.

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
- 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] [-addressing] [-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 segments or interfaces
[**-addressing**]	Get addressing segments of given -of_objects
[**-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
[*patterns*]	Match engine names against patterns Default: *

Categories

IPIntegrator

Description

Get a list of address segments in the current IP Integrator subsystem 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get address segments from all levels of the IP Integrator subsystem design hierarchy, or current instance. Without this argument, the command will only get address segments from the top of the subsystem 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 name. For instance, searching for U1/* searches each level of the hierarchy for objects with U1/ in the name. This may not return the intended results.

NOTE: When used with **-regexp**, the specified search string is matched against the full hierarchical name, and the U1/* search pattern will work as intended

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bd_addr_segs** based on property values on the address segments. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the IP Integrator address segments object, "OFFSET", "RANGE" and "USAGE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 address segments that are assigned to the specified address spaces, as returned by the **get_bd_addr_spaces** command, or of pins and interface pins as returned by **get_bd_pins** and **get_bd_intf_pins**.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

-addressing - (Optional) Get addressing segments of the address spaces specified by the **-of_objects** option. This reports all available segments of the address spaces.

-addressed - (Optional) Get addressed segments of the address spaces specified by the **-of_objects** option. This reports the segments of the address spaces that are utilized.

-addressables - (Optional) Get addressable segments, that are not already occupied by address segments, of the address spaces of the objects specified by the **-of_objects** option.

NOTE: **-addressables**, **-addressed**, and **-addressing** are mutually exclusive, and cannot be used together

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 address segments against the specified patterns. The default pattern is the wildcard '*' which gets a list of all address segments in the current IP subsystem design. More than one pattern can be specified to find multiple address segments based on different search criteria.

NOTE: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the address segments of the specified address spaces:

```
get_bd_addr_segs -of_objects [get_bd_addr_spaces -of_objects \
[get_bd_cells /microblaze_1]]
/microblaze_1/Data/SEG1
/microblaze_1/Data/SEG3
/microblaze_1/Instruction/SEG2
```

NOTE: If there are no objects matching the pattern you will get a warning.

See Also

- create_bd_addr_seg
- get_bd_addr_spaces

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 segments or interfaces
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match engine names against patterns Default: *

Categories

IPIntegrator

Description

Get a list of address spaces in the current IP Integrator subsystem 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get address spaces from all levels of the IP Integrator subsystem design hierarchy, or current instance. Without this argument, the command will only get address spaces 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 name. For instance, searching for U1/* searches each level of the hierarchy for objects with U1/ in the name. This may not return the intended results.

NOTE: When used with **-regexp**, the specified search string is matched against the full hierarchical name, and the U1/* search pattern will work as intended

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bd_addr_spaces** 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 Integrator address space object, "NAME", "RANGE" and "OFFSET" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 address spaces of the specified IP Integrator cell objects, as returned by the **get_bd_cells**.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 address spaces against the specified patterns. The default pattern is the wildcard '*' which gets a list of all address spaces in the current IP subsystem design. More than one pattern can be specified to find multiple address spaces based on different search criteria.

NOTE: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example lists all of the address spaces in the current IP Integrator subsystem design, listing one per line:

```
join [get_bd_addr_spaces] \n
/mdm_1/S_AXI
/microblaze_1/Data
/microblaze_1/Instruction
/microblaze_1_axi_intc/s_axi
/microblaze_1_local_memory/dlmb_bram_if_cntlr/SLMB
/microblaze_1_local_memory/dlmb_bram_if_cntlr/SLMB1
/microblaze_1_local_memory/dlmb_bram_if_cntlr/SLMB2
/microblaze_1_local_memory/dlmb_bram_if_cntlr/SLMB3
/microblaze_1_local_memory/dlmb_bram_if_cntlr/S_AXI_CTRL
/microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB
/microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB1
/microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB2
/microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB3
/microblaze_1_local_memory/ilmb_bram_if_cntlr/S_AXI_CTRL
/microblaze_1_local_memory/lmb_bram/S_1
```

NOTE: If there are no objects matching the pattern you will get a warning.

The following example returns all of the properties attached to the third in a list, or index 2, of all address spaces in the current subsystem design:

```
report_property -all [lindex [get_bd_addr_spaces] 2 ]
Property  Type    Read-only  Visible  Value
CLASS     string  true       true     bd_addr_space
NAME      string  false      true     /microblaze_1/Instruction
OFFSET    string  false      true
PATH      string  true       true     /microblaze_1/Instruction
RANGE     string  false      true     -1
TYPE      string  false      true
```

See Also

- create_bd_addr_seg
- get_bd_addr_segs
- get_bd_cells

get_bd_cells

Get a list of block diagram cells.

Syntax

get_bd_cells [**-regexp**] [**-hierarchical**] [**-filter** *arg*] [**-of_objects** *args*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

List of block diagram 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
[*patterns*]	Match engine names against patterns Default: *

Categories

IPIntegrator

Description

Get a list of cells in the current IP Integrator subsystem design, or current subsystem instance. IP Integrator subsystem cells are either IP cores drawn from the IP Integrator catalog, or hierarchical modules created in the subsystem design with the **create_bd_cell** 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get cells from all levels of the IP Integrator subsystem design hierarchy, or current instance. 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.

NOTE: When used with **-regexp**, the specified search string is matched against the full hierarchical name, and the U1/* search pattern will work as intended

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bd_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 IP Integrator cell object, "VLNV", "TYPE" and "LOCATION" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 subsystem cells that are connected to the specified pin or net objects, as returned by the **get_bd_nets** and **get_bd_pins**, or **get_bd_intf_pins** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 subsystem cells against the specified patterns. The default pattern is the wildcard '*' which gets a list of all cells in the current IP subsystem design. 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 {} to present the list as a single element.

Examples

The following example gets a list of cells that include the specified IP Integrator subsystem pin, and sorts the list to remove duplicate entries:

```
lsort -unique [get_bd_cells -of_objects [get_bd_pins -hierarchical *aclk*]]
```

NOTE: If there are no cells matching the pattern you will get a warning.

The following example gets a list of all cells in all levels of the subsystem design hierarchy, and then filters the list to include only those cells whose name includes the specified text, or hierarchy:

```
get_bd_cells -hierarchical -filter {NAME=~"/newMod1/*"}
```

See Also

- create_bd_cell
- get_bd_intf_pins
- get_bd_nets
- get_bd_pins
- list_property
- report_property

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
[-boundary_type 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
[**-boundary_type**]	Used when source object is on a hierarchical block's interface pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper
[**-of_objects**]	One or a list of cells or interface pins/ports objects. List must be of one object type.
[**-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 interface nets in the current IP Integrator subsystem design that match a specified search pattern. The default command gets a list of all interface nets in the subsystem 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get interface nets from all levels of the IP Integrator subsystem design hierarchy. Without this argument, the command will only get interface nets from the top of the subsystem design, or from the current subsystem instance. 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 name. For instance, searching for /bridge_1/* searches each level of the hierarchy for nets with /bridge_1/ in the name. This may not return the intended results.

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

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bd_intf_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 IP Integrator subsystem nets object, "NAME" and "PATH" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-boundary_type [upper | lower | both] - (Optional) Gets the net 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.

-of_objects *args* - (Optional) Get a list of the nets connected to the specified IP Integrator subsystem cell, pin, or port objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 subsystem interface nets against the specified patterns. The default pattern is the wildcard '*' which returns a list of all interface nets in the current IP Integrator subsystem design. 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 {} to present the list as a single element.

Examples

The following example gets the interface net attached to the specified pin of an IP Integrator hierarchical module, and returns both the net at the level of the hierarchical module, and the net inside the hierarchical module:

```
get_bd_intf_nets -boundary_type both -of_objects [get_bd_pins /newMod1/aclk]
```

NOTE: If there are no interface nets matching the pattern you will get a warning.

The following example returns a list of all interface nets at all levels of the IP Integrator subsystem design hierarchy:

```
get_bd_intf_nets -hierarchical
```

See Also

- connect_debug_port
- get_cells
- get_clocks
- get_pins
- get_ports
- list_property
- report_property

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**]	One or a list of cells, interface nets or pins objects. List must be of one object type.
[**-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 interface pin objects on the current IP subsystem design that match a specified search pattern. The default command gets a list of all interface pins in the subsystem design.

The external connections in an IP Integrator cell, or hierarchical module, are pins and interface pins. The external connections in an IP subsystem design are ports, or interface ports. Use the **get_bd_ports** and **get_bd_intf_ports** commands to select the port objects.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get interface pins from all levels of the IP Integrator subsystem design hierarchy. Without this argument, the command will only get interface pins from the top of the subsystem design, or from the current subsystem instance. 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 name. For instance, searching for /bridge_1/* searches each level of the hierarchy for interface pins with /bridge_1/ in the name. This may not return the intended results.

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

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bd_intf_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 IP integrator interface pins object, "DIR" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 IP subsystem pin, bd_pin, bd_cell, or bd_net.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Integrator subsystem interface pins against the specified patterns. The default pattern is the wildcard '*' which gets a list of all interface pins in the current IP Integrator subsystem design, or instance. 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 {} to present the list as a single element.

Examples

The following example gets a list of interface pins attached to the specified cell:

```
get_bd_intf_pins -of [get_bd_cells new_vidOut_1]
```

NOTE: If there are no interface pins matching the pattern, the tool will return a warning.

The following example gets a list of all interface pins, throughout the hierarchy of the IP Integrator subsystem design, which match the specified name pattern:

```
get_bd_intf_pins -hierarchical m_apb*
```

The following example gets a list of interface pins attached to the specified subsystem net:

```
get_bd_intf_pins -of [get_bd_intf_nets vidout_1_vtg_ce]
```

See Also

- create_bd_net
- create_bd_port
- get_bd_intf_pins
- get_bd_intf_ports
- get_bd_pins
- list_property
- report_property

get_bd_intf_ports

Get a list of intf_ports.

Syntax

get_bd_intf_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**]	One or a list of interface nets objects.
[**-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 interface port objects in the current IP Integrator subsystem design that match a specified search pattern. The default command gets a list of all interface ports in the subsystem design.

The external connections in an IP subsystem design are ports, or interface ports. The external connections in an IP Integrator cell, or hierarchical module, are pins and interface pins. Use the **get_bd_pins** and **get_bd_intf_pins** commands to select the pin objects.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_bd_intf_ports** based on property values on the interface ports. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the IP subsystem interface port object, "DIR", "MODE", and "LOCATION" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 interface ports connected to the specified IP subsystem interface nets returned by **get_bd_intf_nets**.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 interface ports against the specified patterns. The default pattern is the wildcard '*' which gets a list of all interface ports in the subsystem design. More than one pattern can be specified to find multiple interface ports based on different search criteria.

NOTE: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the interface ports in the subsystem design that operate in Master mode:

```
get_bd_intf_ports -filter {MODE=="master"}
```

NOTE: If there are no interface ports matching the pattern, the tool will return a warning.

See Also

- create_bd_intf_net
- create_bd_intf_port
- get_bd_intf_nets
- get_bd_intf_pins
- get_bd_intf_ports
- get_bd_nets
- get_bd_pins
- list_property
- report_property

get_bd_nets

Get a list of nets.

Syntax

```
get_bd_nets [-regexp] [-hierarchical] [-filter arg]
[-boundary_type 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
[**-boundary_type**]	Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper
[**-of_objects**]	One or a list of cells or pins/ports objects. List must be of one object type.
[**-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 nets in the current IP Integrator subsystem design that match a specified search pattern. The default command gets a list of all nets in the subsystem 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get nets from all levels of the IP Integrator subsystem design hierarchy. Without this argument, the command will only get nets from the top of the subsystem design, or from the current subsystem instance. 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 name. For instance, searching for /bridge_1/* searches each level of the hierarchy for nets with /bridge_1/ in the name. This may not return the intended results.

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

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bd_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 IP Integrator subsystem nets object, "NAME" and "PATH" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-boundary_type [upper | lower | both] - (Optional) Gets the net 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.

-of_objects *args* - (Optional) Get a list of the nets connected to the specified cell, pin, or port objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 subsystem nets against the specified patterns. The default pattern is the wildcard '*' which returns a list of all nets in the current IP Integrator subsystem design. 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 {} to present the list as a single element.

Examples

The following example gets the net attached to the specified pin of an IP Integrator hierarchical module, and returns both the net at the level of the hierarchical module, and the net inside the hierarchical module:

```
get_bd_nets -boundary_type both -of_objects [get_bd_pins /newMod1/aclk]
```

NOTE: If there are no nets matching the pattern you will get a warning.

The following example returns a list of all nets at all levels of the IP Integrator subsystem design hierarchy:

```
get_bd_nets -hierarchical
```

See Also

- connect_debug_port
- get_cells
- get_clocks
- get_pins
- get_ports
- list_property
- report_property

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**]	One or a list of cells, nets or interface pins objects. List must be of one object type.
[**-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 pin objects on the current IP subsystem design that match a specified search pattern. The default command gets a list of all pins in the subsystem design.

The external connections in an IP Integrator cell, or hierarchical module, are pins and interface pins. The external connections in an IP subsystem design are ports, or interface ports. Use the **get_bd_ports** and **get_bd_intf_ports** commands to select the port objects.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-hierarchical - (Optional) Get pins from all levels of the IP Integrator subsystem design hierarchy. Without this argument, the command will only get pins from the top of the subsystem design, or from the current subsystem instance. 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 name. For instance, searching for /bridge_1/* searches each level of the hierarchy for pins with /bridge_1/ in the name. This may not return the intended results.

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

-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, "DIR" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 IP subsystem interface pins, bd_intf_pin, bd_ cell, or bd_net.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Integrator subsystem pins against the specified patterns. You must specify the search pattern using the name convention of *bd_cell_name/bd_pin_name*. In this case, using the wildcard '*/*' would return all of the bd_pin objects on all of the bd_cells in the diagram.

NOTE: More than one pattern can be specified to find multiple pins based on different search criteria. You must enclose multiple search patterns in braces {} to present the list as a single element

Examples

The following example gets a list of pins attached to the specified cell:

```
get_bd_pins -of [get_bd_cells new_vidOut_1]
```

NOTE: If there are no pins matching the pattern, the tool will return a warning.

The following example gets a list of all pins, throughout the hierarchy of the IP Integrator subsystem design, which match the specified name pattern:

```
get_bd_pins -hierarchical LMB*
```

The following example gets a list of pins attached to the specified subsystem net:

```
get_bd_pins -of [get_bd_nets vidout_1_vtg_ce]
```

See Also

- create_bd_net
- create_bd_port
- get_bd_intf_pins
- get_bd_intf_ports
- get_bd_pins
- list_property
- report_property

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**]	One or a list of nets objects.
[**-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 port objects in the current IP Integrator subsystem design that match a specified search pattern. The default command gets a list of all ports in the subsystem design.

The external connections in an IP subsystem design are ports, or interface ports. The external connections in an IP Integrator cell, or hierarchical module, are pins and interface pins. Use the **get_bd_pins** and **get_bd_intf_pins** commands to select the pin objects.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_bd_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 IP subsystem port object, "DIR", "MODE", and "LOCATION" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 IP subsystem nets returned by **get_bd_nets**.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 subsystem design. 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 {} to present the list as a single element.

Examples

The following example gets the ports connected to the specified IP subsystem net:

```
get_bd_ports -of_objects [get_bd_nets bridge_1_apb_m]
```

NOTE: If there are no ports matching the pattern, the tool will return a warning.

See Also

- create_bd_net
- create_bd_port
- get_bd_intf_pins
- get_bd_intf_ports
- get_bd_pins
- list_property
- report_property

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_pin

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_pin of these bels, sites, pins, or nets.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match bel_pin against patterns Default: *

Categories

Object, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 bel_pin object, "NAME" and "IS_INVERTED" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_pin objects 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
[**-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 bels of these slr, tiles, sites, cells, clock_regions or nets.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match bels against patterns Default: *

Categories

Object, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 BELs associated with the specified cells, nets, sites, clock_regions, or slrs.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*patterns*]	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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_part_interfaces

Gets the list of interfaces in the board_part that implement busdef specified by VLNV.

Syntax

get_board_part_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
[*patterns*]	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 interfaces defined on the board part in use by the current project or design, as defined by the BOARD_PART property.

The board part provides a representation of the Xilinx FPGA device in the context of the board-level system, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces. The board part file stores information regarding board attributes. The file, called `board_part.xml`, is located in the `data/boards/board_parts` folder of the Vivado Design Suite installation area.

The **current_board_part** command returns the board part in use by the current project. The **get_board_parts** command returns a list of board parts available for use by the current project.

The interfaces available on the **current_board_part** can be used to define the interfaces required in an IP subsystem design, using **create_bd_interface_port** or **create_bd_port**. It can also be used to define the interfaces available in the overall FPGA design using **create_interface** and **create_port**.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_board_part_interfaces** based on property values on the interfaces. You can find the properties on an object with the **report_property** or **list_property** commands. For example:

```
report_property -all [get_board_part_interfaces ddr3*]
```

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 available board part interfaces against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all interfaces available for use in the current project or design. More than one pattern can be specified to find multiple interfaces 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 all interfaces defined on the current board part:

```
join [get_board_part_interfaces] \n
```

See Also

- create_interface
- current_board_part
- current_project
- get_board_part_pins
- get_board_parts

get_board_part_pins

Gets the list of board_part pins object.

Syntax

```
get_board_part_pins [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-quiet] [-verbose] [patterns...]
```

Returns

List of pins in the board_part

Usage

Name	Description
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching
[**-filter**]	Filter list with expression
[**-of_objects**]	Get 'board_part_pin' objects of these types: 'board_part_interface'.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	match board_part 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 pin objects on the board part in use by the current project or design.

The board part provides a representation of the Xilinx FPGA device in the context of the board-level system, and can help define key aspects of the FPGA design, such as clock constraints, I/O port to package pin assignments, and supported interfaces. The board part file stores information regarding board attributes for each supported board. The file, called `board_part.xml`, is located in the `data/boards/board_parts` folder of the Vivado Design Suite installation area.

The **current_board_part** command returns the board part in use by the current project or design. The **get_board_parts** command returns a list of board parts available for use by the current project.

The board part 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 part pins can be scalar or vector, so it is always represented as bitwise.

The board part pins can be used to define and place PORTS in the top-level FPGA design, using the **create_port** and **set_property PACKAGE_PIN** commands.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_board_part_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. For example:

```
report_property [get_board_part_pins RESET]
```

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 pin assignments of the specified board part interface objects for the current board part.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 available board part pins against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all board part pins available for use in the current project or design. More than one pattern can be specified to find multiple interfaces 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 assigns the PACKAGE_PIN and IOSTANDARD properties on the specified port in the current design according to the properties on the leds_8bits pins in the current board:

```
set_property PACKAGE_PIN [get_property LOC \
[get_board_part_pins leds_8bits_TRI_O[1]]] [get_ports LEDS_n[1]]
set_property IOSTANDARD [get_property IOSTANDARD \
[get_board_part_pins leds_8bits_TRI_O[1]]] [get_ports LEDS_n[1]]
```

The following example gets a list of board part pins assigned to the leds_8bits board part interface; stores those pins in a Tcl variable $boardPins, and then prints the LOC property **foreach** of those pins:

```
set boardPins [get_board_part_pins -of \
[get_board_part_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_part
- current_project
- get_board_part_interfaces
- get_board_parts

get_board_parts

Get the list of board_part available in the project.

Syntax

get_board_parts [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns...*]

Returns

List of board_part 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
[*patterns*]	Match Board Part names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

Object, Project, XPS, Board

Description

Gets a list of available board parts, or Target Reference Designs (TRD), available for use by the current project or design.

The board part provides a representation of the Xilinx FPGA device in the context of the board-level system, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces. The board part file stores information regarding board attributes. The file, called `board_part.xml`, is located in the `data/boards/board_parts` folder of the Vivado Design Suite installation area.

The board part in use by the current project is returned by the **current_board_part** command.

The board part is specified as part of a Targeted Reference Design, when the project is defined; or by setting the BOARD_PART 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 part with the **set_property** command, the target part is also changed to match the part required by the specified BOARD_PART property.*

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_board_parts** based on property values on the board 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 board part object, "NAME", "DEVICE", and "FAMILY" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 board parts against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all board parts available for use in the project. More than one pattern can be specified to find multiple boards 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 properties of the specified evaluation board:

```
report_property [get_board_parts -filter {FAMILY==artix7}]
```

The following example returns all board parts matching the specified search patterns:

```
get_board_parts {*av* *kc*}
```

See Also

- current_board_part
- get_board_part_interfaces
- get_board_part_pins
- list_property
- report_property
- set_property

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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching
[**-filter**]	Filter list with expression
[**-of_objects**]	Get 'board_part_pin' objects of these types: 'board_part_interface'.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 assigns the PACKAGE_PIN and IOSTANDARD properties on the specified port in the current design according to the properties on the dip_switch pins in the current board:

```
set_property PACKAGE_PIN [get_property LOC \
[get_board_pins dip_switches_tri_i[0]]] [get_ports {DIPS[0]}]
set_property IOSTANDARD [get_property IOSTANDARD \
[get_board_pins dip_switches_tri_i[0]]] [get_ports {DIPS[0]}]
```

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
[*patterns*]	Match Board names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

Object, Project, XPS, Board

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

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 - (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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 boards against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all boards available for use in the project. More than one pattern can be specified to find multiple boards 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 properties of the specified evaluation board:

```
report_property [get_boards -filter {LIBRARY_NAME==artix7}]
```

The following example returns all boards matching the specified search patterns:

```
get_boards {*ar* *kc*}
```

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
[**-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 cells of these pins, timing paths, nets, bels, clock regions, sites, or drc violations
[**-match_style**]	Style of pattern matching Default: sdc Values: ucf, sdc
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 **-regexp**, 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 "cell" object, "IS_PARTITION", "IS_PRIMITIVE" and "IS_LOC_FIXED" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 pins, timing paths, nets, bels, clock regions, sites or DRC violation objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 uses regular expression to return cells in the BFT example design that match the filter expressions for NAME and REF_NAME. In the first command the NAME is specified in the search pattern, while the second command filters on the NAME property. These commands return the same results:

```
get_cells -regexp -filter { REF_NAME =~ FD.* } .*validFor.*
get_cells -regexp -filter { NAME =~ .*validFor.* && REF_NAME =~ FD.* }
```

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.

The following example runs the **report_drc** command on the current design, and returns any cells associated with DRC violations:

```
report_drc -name drc_1
get_cells -of_objects [get_drc_violations]
```

See Also

- get_lib_cells
- get_nets
- get_pins
- list_property
- report_drc
- report_property

get_cfgmem_parts

Get a list of cfgmem_parts available in the software.

Syntax

get_cfgmem_parts [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

List of cfgmem_part 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 the 'cfgmem_part' objects against patterns. Default: *

Categories

Hardware, Object

Description

Get a list of configuration flash memory devices supported by the Vivado Design Suite.

Xilinx FPGAs can be configured from an external nonvolatile memory device, or they can be configured by an external smart source, such as a microprocessor, DSP processor, microcontroller, PC, or board tester. The two configuration datapaths include a serial datapath that is used to minimize the device pin requirements for configuration, and a parallel 8-bit, 16-bit, or 32-bit datapath used for higher performance or link to industry-standard interfaces ideal for external data sources like processors, or x8- or x16-parallel flash memory.

The process whereby the design specific data is loaded or programmed into the Xilinx® FPGA is called configuration. The **create_hw_cfgmem** command defines a flash memory device used for configuring and booting the hardware device.

After the hw_cfgmem object is created, and associated with the hw_device, the configuration memory can be programmed with the bitstream and other data from a memory configuration file created with the **write_cfgmem** command. The hw_cfgmem object is programmed using the **program_hw_cfgmem** command.

The configuration memory part can be used to define the hardware configuration memory in the Hardware Manager of the Vivado Design Suite, to enable programming and debugging the design in the FPGA hardware. Use the **create_hw_cfgmem** command to define the configuration memory for use with the current hardware device.

This command returns a list of cfgmem_part objects that match the specified search criteria, or returns and error if it fails.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_cfgmem_parts** 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 "cfgmem_part" object, "COMPATIBLE_PARTS", "DATA_WIDTH", and "MEM_DENSITY" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 cfgmem_parts against the specified patterns. The default pattern is the wildcard '*' which gets a list of all cfgmem_parts currently available in the Vivado Design Suite.

Example

The following example gets the PART for the current_hw_device, and filters the list of available configuration memory devices to include only those compatible with the target part with a specific amount of memory:

```
set devPart [get_property PART [current_hw_device]]
set cfgParts [get_cfgmem_parts -filter \
   "COMPATIBLE_PARTS =~ [concat "{*$devPart*}"] && MEM_DENSITY > 64 "]
```

NOTE: The **concat** command builds the substring "{*xc7k325t*}" required to locate the device part in the COMPATIBLE_PARTS property string

See Also

- create_hw_cfgmem
- current_hw_device
- delete_hw_cfgmem
- get_property
- program_hw_cfgmem
- set_property
- write_cfgmem

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
[**-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 clock_regions of these tiles, sites, slrs, or cells
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 clock regions of the SLRs that the clock region is part of using the **get_slrs** command. Or get the clock_regions that the specified tiles, device sites, or cells are assigned to.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

The following example returns the clock regions that the specified ILA debug core trigger is assigned to, or placed in:

```
get_clock_regions -of_objects [get_cells -hierarchical basic_trigger*]
```

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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-filter**]	Filter list with expression
[**-of_objects**]	Get clocks of these pins, nets, or cells
[**-match_style**]	Style of pattern matching, valid values are ucf, sdc Default: sdc
[**-include_generated_clocks**]	Include auto-inferred/generated_clocks also.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 an 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 cell, pin, or net objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-filter**]	Filter list with expression
[**-of_objects**]	Get cores of these debug ports or nets
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match debug cores against patterns Default: *

Categories

Object, Debug, XDC

Description

Gets a list of Vivado lab tools 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_v3) is added to the project, it is contained within a Debug Hub core (labtools_xsdbmasterlib_v2), and includes a CLK port and a PROBE port 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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 ILA debug cores associated with the specified debug ports, or nets.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Vivado lab tools debug cores in the current project:

```
get_debug_cores
```

NOTE: A Debug Hub 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]
```

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
[**-filter**]	Filter list with expression
[**-of_objects**]	Get ports of these debug cores
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match debug ports against patterns Default: *

Categories

Object, Debug, XDC

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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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/PROBE1]
```

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_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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-of_objects**]	Get 'drc_check' 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
[**-abbrev**]	Get the largest ID for this abbrev
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match the 'drc_check' 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 [**-of_objects** *args*] [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Drc_ruledeck

Usage

Name	Description
[**-of_objects**]	Get 'drc_ruledeck' objects of these types: 'drc_check'.
[**-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

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_violations

Get a list of DRC violations from a previous report_drc run.

Syntax

get_drc_violations [**-name** *arg*] [**-regexp**] [**-filter** *arg*] [**-nocase**] [**-quiet**] [**-verbose**] [*patterns*]

Returns

List of DRC violation 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_violations 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 found in the design when the report_drc command is run. Violation objects are created at the time DRC is run, either by the internal design rule checks provided by the Vivado tools, or created by the **create_drc_violation** command in user-defined DRC checks. The properties of individual violation objects can be queried using **report_property** or **list_property** commands for details of the violation.

Violation objects are associated with the cells, nets, pins, or ports in the current design, or sites on the current device. The design objects associated with a DRC violation object can be obtained using the **-of_objects** option of the appropriate **get_*** command, such as **get_cells**, or **get_nets** 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

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_violations** based on property values on the violations. You can find the properties on an object with the **report_property** or **list_property** commands.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_violations
```

The following example gets the properties of the specified DRC violation:

```
report_property [lindex [get_drc_violations] 0]
```

The following example returns the list of violations in the specified DRC report of the current design, and then returns the ports associated with any violations of the unspecified I/O Standard rule (NSTD):

```
get_drc_violations -name drc_1
get_ports -of_objects [get_drc_violations -name drc_1 NSTD*]
```

See Also

- create_drc_check
- create_drc_violation
- get_cells
- get_nets
- get_pins
- get_ports
- get_sites
- report_drc

get_files

Get a list of source files.

Syntax

```
get_files [-regexp] [-nocase] [-filter arg] [-compile_order arg]
[-used_in arg] [-all] [-of_objects args] [-quiet] [-verbose]
[patterns]
```

Returns

List of file 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
[**-compile_order**]	Get files by type and in compilation order (must use with -used_in)
[**-used_in**]	Get files used in a specific step (must use with -compile_order)
[**-all**]	Include all internal files.
[**-of_objects**]	Get 'file' objects of these types: 'file fileset ip'.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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.

The **get_files** command returns a machine readable list of files in the project, in a design, or in a sub-design such as an IP core or block design. You can filter the results returned by **get_files** using one of the command arguments such as **-of_objects**, **-compile_order**, **-used_in**, and **-filter**.

The **-compile_order** and **-used_in** options must be used together to return a list of files from the sources or constraints filesets used in synthesis, simulation, **OR** implementation, sorted according to the order of evaluation by the Vivado tools. The **-compile_order** and **-used_in** options use complex file ordering rules that can change based on header files, include files, or other language options. You can also filter files returned by **get_files** according to the USED_IN property, using the **-filter** option instead of the **-used_in** option.

You can use the **report_compile_order** command to generate a report of all files in the sources or constraints filesets, used in synthesis, simulation, AND implementation, sorted into different sections.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 files that are associated with the specified fileset objects. The default is to search all filesets. When **-compile_order** and **-used_in** are specified, the **-of_objects** switch will only accept a single fileset, or a single sub-design such as an IP core, Block Design, or DSP design. A sub-design is also known as a composite file.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

-compile_order [sources | constraints] - (Optional) Returns the source design files, or the constraint files sorted according to the order of compilation by the Vivado Design Suite.

NOTE: This option must be used with the **-used_in** option. When specified, the **-of_objects** switch will only accept a single fileset or sub-design.

-used_in *arg* - (Optional) Accepts one of the enumerated values of the **USED_IN** property for files, and returns files matching the specified value. Valid values for this option include the following: synthesis, simulation, or implementation. Refer to the *Vivado Design Suite Properties Reference Guide (UG912)* for information on the USED_IN property and its supported values.

NOTE: This option must be used with the **-compile_order** option. When specified, the **-of_objects** switch will only accept a single fileset or sub-design.

-all - (Optional) Returns all of the files in the design, including internal files in support of IP and other objects.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 VHDL files in the design that are used in simulation:

```
get_files -filter {FILE_TYPE == VHDL && USED_IN =~ simulation}
```

This example returns the design source files that are used in simulation:

```
get_files -compile_order sources -used_in simulation
```

This example gets a list of files associated with the specified IP core composite file (ip.xci), from the `sources_1` fileset that are used in synthesis:

```
get_files -compile_order sources -used_in synthesis -of [get_files ip.xci]
```

The following example gets a list of the files found in the sources_1 and constrs_1 filesets:

```
get_files -of [get_filesets {sources_1 constrs_1}]
```

NOTE: If there are no files matching the pattern you will get a warning.

See Also

- report_compile_order
- 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
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-filter**]	Filter list with expression
[**-of_objects**]	Get generated clocks of these pins, ports or nets
[**-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 generated clock names against patterns Default: *

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 an 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 generated clocks connected to the specified port, pin, or net objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_highlighted_objects

Get highlighted objects.

Syntax

```
get_highlighted_objects [-color_index arg] [-rgb args] [-color arg]
[-quiet] [-verbose]
```

Returns

List of highlighted objects

Usage

Name	Description
[**-color_index**]	Color index
[**-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

Categories

Object, GUIControl

Description

Gets the highlighted objects in the current design open in the Vivado IDE. Objects can be highlighted with the **highlight_objects** command.

You can get all highlighted objects in the design, or specify highlighted objects by color, by color index, or by RGB value.

NOTE: This Tcl command works only when Vivado is run in GUI mode

Arguments

-color_index arg - (Optional) Valid values are integers from 1 to 19. Specifies the color index used 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 used for highlighting objects 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 used 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 all the highlighted objects in the current design:

```
get_highlighted_objects
```

The following example gets the object in the current design that are highlighted in the specified color:

```
get_highlighted_objects -color cyan
```

See Also

- get_marked_objects
- get_selected_objects
- highlight_objects
- mark_objects
- select_objects

get_hw_axi_txns

Get a list of hardware AXI transactions.

Syntax

get_hw_axi_txns [**-of_objects** *args*] [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Hw_axi_txns

Usage

Name	Description
[**-of_objects**]	Get 'hw_axi_txn' objects of these types: 'hw_axi'.
[**-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_axi_txn' objects against patterns. Default: *

Categories

Hardware, Object

get_hw_axis

Get a list of hardware AXI objects.

Syntax

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

Returns

Hw_axi

Usage

Name	Description
[**-of_objects**]	Get 'hw_axi' 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_axi' objects against patterns. Default: *

Categories

Hardware, Object

Description

Returns the JTAG to AXI Master core objects defined on the current hardware device.

The JTAG to AXI Master core, or hw_axi object, is a customizable IP core that works as an AXI Master to drive AXI transactions and drive AXI signals on the Xilinx FPGA device, hw_device object. The AXI Master core supports AXI4 interfaces and AXI-Lite protocol. The width of AXI data bus is configurable. The AXI core can drive AXI4-Lite or AXI4 Memory mapped Slave through an AXI4 interconnect. The core can also be connected to interconnect as the master. Detailed documentation on the VIO core can be found in the LogiCORE IP JTAG to AXI Master Product Guide (PG174).

The JTAG to AXI Master core must be instantiated in the RTL code, from the Xilinx IP catalog. AXI cores can be found in the synthesized netlist design using the get_debug_cores command. These are not the hardware AXI Master core objects, hw_axi, found in the Hardware Manager feature of the Vivado Design Suite, though they are related.

This command returns a list of AXI Master core objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the AXI Master cores of the specified hardware devices. The devices must be specified as objects using the **get_hw_devices** or the **current_hw_device** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_axis** based on property values on the debug cores. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_axi" object, "NAME" and "PROTOCOL" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard “*” character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the “RESET” substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_axi objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_axis available on the current hardware device.

Example

The following example gets the ILA debug cores defined on the current hardware device:

```
get_hw_axis -of_objects [current_hw_device]
```

See Also

- connect_hw_server
- current_hw_device
- program_hw_devices
- set_property

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

Description

Returns the hardware devices on the open target of a connected hardware server.

Each hardware target can have one or more Xilinx FPGA devices to program, or to use for debugging purposes. The current device is specified or returned by the **current_hw_device** command.

This command returns a list of available hardware device objects, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the hardware devices of the specified hardware targets. The targets must be specified as objects using the **get_hw_targets** or the **current_hw_target** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_devices** 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 "hw_device" object, "NAME" and "PART" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_devices against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_devices available on the connected hardware server.

Example

The following example gets the list of available hardware device objects on the current hardware target:

```
get_hw_devices -of_objects [current_hw_target]
```

See Also

- connect_hw_server
- current_hw_device
- current_hw_target
- get_hw_targets
- open_hw_target

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

Description

Returns the ILA data objects in the Vivado logic analyzer.

The **upload_hw_ila_data** command creates a hw_ila_data object in the process of moving the captured data from the ILA debug core, hw_ila, on the physical FPGA device, hw_device. The **read_hw_ila_data** command can also create a hw_ila_data object when reading an ILA data file from disk.

The hw_ila_data object can be viewed in the waveform viewer of the Vivado logic analyzer by using the **display_hw_ila_data** command, and can be written to disk using the **write_hw_ila_data** command.

This command returns a list of available hardware ILA data objects, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the hw_ila_data objects of the specified hw_ila debug objects. The hw_ila cores must be specified as objects using the **get_hw_ilas** or the **current_hw_ila** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_ila_datas** based on property values on the hw_ila_data objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_ila_data" object, "NAME" and "TIMESTAMP" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_ila_data objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_ila_data objects available in the Vivado logic analyzer.

Example

The following example gets the data objects for all the hardware ILA debug cores on the current device:

```
get_hw_ila_datas -of_objects [get_hw_ilas]
```

See Also

- current_hw_device
- current_hw_ila
- current_hw_ila_data
- display_hw_ila_data
- get_hw_devices
- get_hw_ilas
- read_hw_ila_data
- upload_hw_ila_data
- write_hw_ila_data

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
[*patterns*]	Match the 'hw_ila' objects against patterns. Default: *

Categories

Hardware, Object

Description

Returns the ILA debug core objects defined on the current hardware device.

The Integrated Logic Analyzer (ILA) debug core lets you perform in-system debug of implemented designs, or design bitstreams, on a programmed Xilinx FPGA device. The ILA core includes many advanced features of modern logic analyzers, including boolean trigger equations, and edge transition triggers. You can use the ILA core to probe specific signals of the design, to trigger on programmed hardware events, and capture data from the Xilinx FPGA device in real-time. Refer to *LogiCORE IP Integrated Logic Analyzer (PG172)* for details of the ILA core.

You can add ILA debug cores into the RTL source files of a design, or in the synthesized netlist using the **create_debug_core** command. Refer to the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)* for more information on adding debug cores and signal probes to the design. You can get the debug cores in the synthesized netlist by using the **get_debug_core** commands. However, though they are related, the design debug cores are separate from the hardware debug cores found in the Hardware Manager feature of the Vivado Design Suite.

Debug cores and probes are written to the probes file (.ltx) using the **write_probes_file** command and associated with the hardware device, along with the bitstream file (.bit), using the PROBES.FILE and PROGRAM.FILE properties of the hw_device object. The hardware device is programmed with this information using the **program_hw_devices** command.

The steps to debug your design in hardware using an ILA debug core are:

1. Connect to the hardware server and target using **connect_hw_server** and **open_hw_target**.
2. Program the FPGA device with the bitstream (.bit) and probes (.ltx) files using **program_hw_devices**.
3. Set up the ILA debug core trigger and capture controls using **set_property** to set properties of the ILA.
4. Arm the ILA debug core trigger using **run_hw_ila**.
5. View the captured data from the ILA debug core in the Waveform window of the Vivado logic analyzer feature using **display_hw_ila_data**.
6. Optionally use the VIO debug core to drive control signals and/or view design status signals. See **get_hw_vios** for more information.
7. Optionally use the JTAG-to-AXI Master debug core to run transactions to interact with various AXI slave cores in your design. See **get_hw_axis** for more information.

This command returns a list of ILA debug core objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the ILA debug cores of the specified hardware devices. The devices must be specified as objects using the **get_hw_devices** or the **current_hw_device** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_ilas** 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 "hw_ila" object, "NAME" and "CONTROL.DATA_DEPTH" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_ilas against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_ilas available on the current hardware device.

Example

The following example gets the ILA debug cores defined on the current hardware device:

```
get_hw_ilas -of_objects [current_hw_device]
```

See Also

- connect_hw_server
- create_debug_core
- current_hw_device
- current_hw_ila
- display_hw_ila_data
- get_hw_axis
- get_hw_devices
- get_hw_ila_datas
- get_hw_vios
- open_hw_target
- program_hw_devices
- run_hw_ila
- set_property

get_hw_migs

Get list of hardware Migs cores.

Syntax

get_hw_migs [**-of_objects** *args*] [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Hardware migs cores

Usage

Name	Description
[**-of_objects**]	Get 'hw_mig' objects of these types: 'hw_server hw_target 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_mig' 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
[*patterns*]	Match the 'hw_probe' objects against patterns. Default: *

Categories

Hardware, Object

Description

Returns the hw_probe objects in the Hardware Manager that are defined on signals in the design, or that are assigned to the specified ILA or VIO debug cores.

You can add ILA and VIO debug cores in the RTL source files of a design by customizing the core from the IP catalog, or add ILA debug cores into the synthesized netlist using the **create_debug_core** command.

Signals in the design can be probed to monitor signal values and track hardware events on the FPGA device. Debug probes can be added to ILA debug cores in the synthesized netlist design using the **create_debug_probe** command, and connected to signals in the design using **connect_debug_probe**. Probes can only be added to VIO debug cores when the IP core is customized, or re-customized, from the IP catalog, and signals connected to it in the RTL design. Refer to the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)* for more information on adding ILA and VIO debug cores and signal probes to the design.

Debug cores and probes are written to a probes file (.ltx) with write_debug_probes, and associated with the hardware device, along with the bitstream file (.bit), using the PROBES.FILE and PROGRAM.FILE properties of the hw_device object. The hardware device is programmed with this information using the **program_hw_device** command.

This command returns a list of debug probe objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the debug probes assigned to the specified ILA or VIO debug cores. The ILA or VIO cores must be specified as hardware objects using the **get_hw_ilas**, **current_hw_ila**, **get_hw_vios**, or **current_hw_vio** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_probes** 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 "hw_probes" object, "NAME" and "SIGNAL" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_probes against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_probes assigned on the current device.

Example

The following example gets probes assigned to the current ILA debug core:

```
get_hw_probes -of_object [current_hw_ila]
```

See Also

- connect_debug_port
- create_debug_core
- create_debug_port
- current_hw_device
- current_hw_ila
- get_hw_devices
- get_hw_ilas
- get_hw_vios

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

Description



IMPORTANT: *You must use the **open_hw** command to open the Hardware Manager in the Vivado Design Suite before using this command.*

This command returns hardware server objects that are connected to the current instance of the Vivado Design Suite through the use of the **connect_hw_server** command.

Hardware servers are instances of the Vivado tools CSE server (**vcse_server**) application running remotely, or on the local machine. The hardware server manages connections to a hardware target, for instance a hardware board containing a JTAG chain of one or more Xilinx FPGA devices to be used for programming and debugging your FPGA design.

This command returns a list of connected hardware server objects.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_servers** based on property values on the hw_servers. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_server" object, "HOST", "NAME" and "PORT" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_servers against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_servers currently connected.

Example

The following example returns a list of all hw_servers connected to the Vivado Design Suite:

```
get_hw_servers
```

See Also

- connect_hw_server
- current_hw_server
- disconnect_hw_server
- refresh_hw_server

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

Description

Returns the QPLL objects, hw_sio_commons, defined on the IBERT debug core on the current hardware device.

For each serial transceiver channel, there is a ring PLL called Channel PLL (CPLL). In Xilinx UltraScale and 7 series FPGAs, the high-speed transceivers have an additional common or shared PLL per quad, or Quad PLL (QPLL). This QPLL is a shared LC PLL to support high speed, high performance, and low power multi-lane applications.

On the device, the GTXE2_CHANNEL component has the serial transceiver and CPLL units and the GTXE2_COMMON has the QPLL unit.

This command returns a list of QPLL objects on the device as hw_sio_common objects, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the QPLL objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, or hw_sio_pll. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_commons** based on property values on the debug cores. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_common" object, "NAME" and "DISPLAY_NAME" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_commons against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_commons available on the current hardware device.

Example

The following example gets the ILA debug cores defined on the current hardware device:

```
get_hw_sio_commons -of [get_hw_sio_iberts]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_targets
- report_property

get_hw_sio_gtgroups

Get list of hardware SIO GT groups.

Syntax

```
get_hw_sio_gtgroups [-of_objects args] [-regexp] [-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'.
[**-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_gtgroup' objects against patterns. Default: *

Categories

Hardware, Object

Description

Returns the groups of GTs, hw_sio_gtgroups, as they relate to the quads that are in use on the IBERT debug core on the current hardware device.

GT groups, relate to the IO Banks on the hardware device, with the number of available banks and GT pins determined by the target device. On the Kintex-7 xc7k325 part, for example, there are four quads, each containing four differential GT pins, and two differential REFCLK pins. Each GT pin has its own transmitter, TX, and receiver, RX. Quads can also include one shared PLL per quad, or Quad PLL. The quads are defined on the IBERT debug core, and can be customized with a number of user settings when the IBERT is added into the RTL design. Refer to the *LogiCORE IP Integrated Bit Error Ratio Tester (IBERT) for 7 Series GTX Transceivers v3.0 (PG132)*or to *7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)* or *UltraScale Architecture GTH Transceivers User Guide (UG576)* for more information.

This command returns a list of GT group objects on the IBERT debug core, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the hw_sio_gtgroup objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, hw_sio_common, hw_sio_pll, hw_sio_gt, hw_sio_tx, or hw_sio_rx. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_gtgroups** based on property values on the quads. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_gtgroup" object, "DISPLAY_NAME" and "GT_TYPE" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_gtgroup objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_gtgroups available on the current hardware device.

Example

The following example gets the ILA debug cores defined on the current hardware device:

```
get_hw_sio_gtgroups -of [get_hw_sio_gts *MGT_X0Y9]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property

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

Description

Returns the Gigabit Transceiver objects (GTs), hw_sio_gt, that are in use on the IBERT debug core on the current hardware device.

The customizable LogiCORE™ IP Integrated Bit Error Ratio Tester (IBERT) core for Xilinx FPGAs is designed for evaluating and monitoring the Gigabit Transceivers (GTs). The IBERT core enables in-system serial I/O validation and debug, letting you measure and optimize your high-speed serial I/O links in your FPGA-based system. Refer to the *LogiCORE IP Integrated Bit Error Ratio Tester (IBERT) for 7 Series GTX Transceivers v3.0 (PG132)*or to *7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)* or *UltraScale Architecture GTH Transceivers User Guide (UG576)* for more information.

Using the IBERT debug core you can configure and tune the GT transmitters and receivers through the Dynamic Reconfiguration Port (DRP) port of the GTX transceiver. This lets you change property settings on the GTs, as well as registers that control the values on the ports.

This command returns a list of GT objects on the IBERT debug core, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the GT objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, hw_sio_pll, hw_sio_tx, hw_sio_rx, or hw_sio_link. The objects must be specified with an appropriate **get_hw_*** command.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_gts** based on property values on the GTs. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_gt" object, "NAME" and "GT_TYPE" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_gt objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_hw_sio_gts available on the current debug core.

Example

The following example gets the GTs assigned to the defined communication links on the IBERT debug core:

```
get_hw_sio_gts -of_objects [get_hw_sio_links]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property

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

Description

Returns the Integrated Bit Error Ratio Tester (IBERT) debug core objects, hw_sio_ibert, defined on the current hardware device.

The customizable LogiCORE™ IP Integrated Bit Error Ratio Tester (IBERT) core for Xilinx FPGAs is designed for evaluating and monitoring the Gigabit Transceivers (GTs). The IBERT core enables in-system serial I/O validation and debug, letting you measure and optimize your high-speed serial I/O links in your FPGA-based system. Refer to the *LogiCORE IP Integrated Bit Error Ratio Tester (IBERT) for 7 Series GTX Transceivers v3.0 (PG132)* for more information.

The IBERT debug core lets you configure and control the major features of GTs on the device, including:

- TX pre-emphasis and post-emphasis
- TX differential swing
- RX equalization
- Decision Feedback Equalizer (DFE)
- Phase-Locked Loop (PLL) divider settings

You can use the IBERT core when you are interested in addressing a range of in-system debug and validation problems; from simple clocking and connectivity issues to complex margin analysis and channel optimization issues.

This command returns a list of IBERT debug core objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the IBERT cores of the specified hw_server, hw_target, hw_device, hw_sio_gt, hw_sio_common, hw_sio_pll, hw_sio_tx, hw_sio_rx, or hw_sio_link objects. The objects must be specified by an appropriate **get_hw_*** command.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_iberts** based on property values on the debug cores. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_ibert" object, "NAME" and "CORE_REFRESH_RATE_MS" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_ibert objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_ibert objects available on the current hardware device.

Example

The following example gets the ILA debug cores defined on the current hardware device:

```
get_hw_sio_iberts -of_objects [current_hw_device]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_plls
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property

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
[*patterns*]	Match the 'hw_sio_linkgroup' objects against patterns. Default: *

Categories

Hardware, Object

Description

Returns the defined groups, or associations of communication links between TX and RX objects on the GTs of the IBERT debug core defined on the current hardware device.

Vivado Serial I/O analyzer is a link-based analyzer. The links define the communication paths and protocols between transmitters and receivers of the GigaBit transceivers on the device. Link groups, or hw_sio_linkgroup objects, let you associate links into related groups, to collectively configure properties and run scans.

This command returns a list of linkgroups on the IBERT debug core, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the linkgroup objects of the specified hw_sio_links. The links must be specified as objects using the **get_hw_sio_links command**.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_linkgroups** based on property values on the link groups. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_linkgroup" object, "NAME" and "DESCRIPTION" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_linkgroup objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_linkgroups available on the current hardware device.

Example

The following example gets all of the link groups defined on the IBERT debug core on the current hardware device:

```
get_hw_sio_linkgroups
```

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property

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

Description

Returns the communication links between TX and RX objects on the GTs of the IBERT debug core defined on the current hardware device.

Vivado Serial I/O analyzer is a link-based analyzer, which lets you link between any transmitter and receiver within the IBERT design. The links define the communication paths and protocols between transmitters and receivers of the GigaBit transceivers on the device. You can configure the links by using the set_property command to specify property values on the link object.

This command returns a list of link objects on the IBERT debug core, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the hw_sio_link objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, hw_sio_gt, hw_sio_tx, hw_sio_rx, or hw_sio_linkgroup. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_links** based on property values on the link objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_link" object, "DESCRIPTION", "TX_ENDPOINT", and "RX_ENDPOINT" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_links against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_link objects available on the current hardware device.

Example

The following example returns links based on the DESCRIPTION property that can be specified when the link is created:

```
get_hw_sio_links -filter {DESCRIPTION == "Link4" || DESCRIPTION == "Link2"}
```

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property

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

Description

Returns the PLL objects, hw_sio_pll, defined on the IBERT debug core on the current hardware device.

For each serial transceiver channel, there is a ring PLL called Channel PLL (CPLL). In Xilinx UltraScale and 7 series FPGAs, the GTX has an additional shared PLL per quad, or Quad PLL (QPLL). This QPLL is a shared LC PLL to support high speed, high performance, and low power multi-lane applications.

On the device, the GTXE2_CHANNEL component has the serial transceiver and CPLL units and the GTXE2_COMMON has the QPLL unit.

This command returns a list of all PLL objects, both CPLLs and QPLLs on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the PLL objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, hw_sio_gt, or hw_sio_common objects. The objects must be specified using the appropriate **get_hw_*** command for the object.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_plls** based on property values on the PLLs. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_pll" object, "NAME" and "PARENT" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_pll objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_plls available on the current hardware device.

Example

The following example returns a list of the PLL objects on the IBERT debug core:

```
join [get_hw_sio_plls -of [get_hw_sio_iberts] ] \n
```

This example returns the PLL objects on the hw_sio_commons of the IBERT debug core:

```
join [get_hw_sio_plls -of [get_hw_sio_commons ]] \n
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_iberts
- get_hw_sio_rxs
- get_hw_sio_txs
- get_hw_targets
- report_property

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

Description

Returns the receiver objects, hw_sio_rx, of the Gigabit Transceivers (GTs) that are in use on the IBERT debug core on the current hardware device.

On the hardware device, each GT includes an independent receiver, which consists of a PCS and a PMA. High-speed serial data flows from traces on the board into the PMA of the GTX/GTH transceiver RX, into the PCS, and finally into the FPGA logic.

This command returns a list of receiver objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the receiver objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, hw_sio_gt, or hw_sio_link. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_rxs** based on property values on the receivers. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_rx" object, "DISPLAY_NAME" and "RX_DATA_WIDTH" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_rx objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_rxs available on the current hardware device.

Example

The following example gets the receiver objects associated with the defined communication links on the IBERT debug core:

```
get_hw_sio_rxs -of [get_hw_sio_links]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_sio_txs
- get_hw_targets
- report_property

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 hw_sio_link hw_sio_sweep'.
[**-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

Description

Returns serial I/O analyzer scan objects for the IBERT debug core.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized Eye Scan hardware of Xilinx UltraScale devices or 7 Series FPGAs. The Vivado serial I/O analyzer feature lets you to create, run, and save link scans.

This command returns one or more hw_sio_scan objects, or returns an error if he command fails.

Arguments

-of_objects *arg* - (Optional) Return the hw_sio_scan objects of the specified hw_sio_rx, hw_sio_link, or hw_sio_sweep. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_scans** based on property values on the scans. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_scan" object, "NAME" and "DESCRIPTION" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_scan objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_scans available on the IBERT debug core.

Example

The following example gets the serial I/O analyzer scans:

```
get_hw_sio_scans
```

See Also

- create_hw_sio_scan
- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_scans
- get_hw_sio_sweeps
- remove_hw_sio_scan
- remove_hw_sio_sweep
- run_hw_sio_scan
- run_hw_sio_sweep
- stop_hw_sio_scan
- stop_hw_sio_sweep
- wait_on_hw_sio_scan
- wait_on_hw_sio_sweep
- write_hw_sio_scan
- write_hw_sio_sweep

get_hw_sio_sweeps

Get list of hardware SIO sweeps.

Syntax

get_hw_sio_sweeps [**-of_objects** *args*] [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Hardware SIO sweeps

Usage

Name	Description
[**-of_objects**]	Get 'hw_sio_sweep' objects of these types: 'hw_sio_link hw_sio_scan'.
[**-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_sweep' objects against patterns. Default: *

Categories

Hardware, Object

Description

Return the serial I/O analyzer link sweep objects defined on the IBERT debug core.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized features of Xilinx UltraScale devices or 7 Series FPGAs. It can also be helpful to run multiple scans on a the link with different configuration settings for the GTs. This can help you determine which settings are best for your design. The Vivado serial I/O analyzer feature enables you to define, run, and save link sweeps, or collections of link scans run across a range of values.

This command returns a link sweep object that you can use with the **run_hw_sio_sweep** command to run analysis on the specified links, or GT receivers. You can also save the sweep scan to disk using the **write_hw_sio_sweep** command.

You can remove the created scan object using **remove_hw_sio_sweep**.

This command returns one or more hw_sio_sweep objects, or returns an error if the command fails.

Arguments

-of_objects *arg* - (Optional) Return the serial I/O analyzer sweep object of the specified hw_sio_link or hw_sio_scan. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_sweeps based on property values on the sweep scans. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_sweep" object, "NAME" and "DESCRIPTION" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard “*” character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the “RESET” substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_sweep objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_sweeps available on the current hardware device.

Example

The following example gets the sweep scans on the IBERT debug core:

```
get_hw_sio_sweeps
```

See Also

- create_hw_sio_scan
- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_scans
- get_hw_sio_sweeps
- remove_hw_sio_scan
- remove_hw_sio_sweep
- run_hw_sio_scan
- run_hw_sio_sweep
- stop_hw_sio_scan
- stop_hw_sio_sweep
- wait_on_hw_sio_scan
- wait_on_hw_sio_sweep
- write_hw_sio_scan
- write_hw_sio_sweep

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

Description

Returns the transmitter objects, hw_sio_tx, of the Gigabit Transceivers (GTs) that are in use on the IBERT debug core on the current hardware device.

On the hardware device, each GT includes an independent transmitter, which consists of a PCS and a PMA. Parallel data flows from the device logic into the FPGA TX interface, through the PCS and PMA, and then out the TX driver as high-speed serial data.

This command returns a list of transmitter objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the transmitter objects of the specified hw_server, hw_target, hw_device, hw_sio_ibert, hw_sio_gt, or hw_sio_link. The objects must be specified using the appropriate **get_hw_* command.**

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_sio_txs** based on property values on the transmitters. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_sio_tx" object, "DISPLAY_NAME" and "TXUSRCLK_FREQ" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_sio_tx objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_sio_txs available on the current hardware device.

Example

The following example returns a list of the transmitters on the current device:

```
join [get_hw_sio_txs] \n
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_sio_rxs
- get_hw_targets
- report_property

get_hw_sysmon_reg

Get the system monitor register value.

Syntax

`get_hw_sysmon_reg` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`hw_sysmon hexaddress`*

Returns

Register value in Hex

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_sysmon	hw_sysmon object
hexaddress	Hex address to read from

Categories

Hardware

get_hw_sysmons

Get list of hardware SysMons.

Syntax

get_hw_sysmons [**-of_objects** *args*] [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Hardware sysmons

Usage

Name	Description
[**-of_objects**]	Get 'hw_sysmon' objects of these types: 'hw_server hw_target 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_sysmon' 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
[*patterns*]	Match the 'hw_target' objects against patterns. Default: *

Categories

Hardware, Object

Description

Returns the available hardware targets of the connected hardware servers.

The hardware target is a system board containing a JTAG chain of one or more Xilinx FPGA devices that you can program with a bitstream file, or use to debug your design. Connections between hardware targets on the system board and the Vivado Design Suite are managed by the **vcse_server** application. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a list of supported JTAG download cables and devices.

Use the **open_hw_target** command to open a connection to one of the available hardware targets. The open target is automatically defined as the current hardware target. Alternatively, you can define the current target with the **current_hw_target** command, and then open a connection to the current target. The Vivado Design Suite directs programming and debug commands to the open target through the hardware server connection.

This command returns a list of available hardware targets through all connected hardware servers, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the hardware targets of the specified hardware server. The hardware server must be specified as a hw_server object using the **get_hw_servers** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_targets** based on property values on the targets. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_target" object, "NAME" and "IS_OPENED" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_targets against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_targets available on the connected hardware server.

Example

The following example returns the available hardware targets on the currently connected hardware servers:

```
get_hw_targets
```

See Also

- connect_hw_server
- current_hw_target
- get_hw_targets
- get_hw_servers
- open_hw_target

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

Description

Returns the Virtual Input/Output (VIO) debug core objects that are defined on the current hardware device, hw_device.

The Virtual Input/Output (VIO) debug core can both monitor and drive internal signals on a programmed Xilinx FPGA device in real time. In the absence of physical access to the target hardware, you can use this debug feature to drive and monitor signals that are present on the physical device.

The VIO core has hardware probes, hw_probe objects, to monitor and drive specific signals on the design. Input probes monitor signals as inputs to the VIO core. Output probes drive signals to specified values from the VIO core. Values on the debug core are driven onto the signals at the probe using the **commit_hw_vio** command.

The VIO debug core needs to be instantiated in the RTL code, therefore you need to know what nets you want monitor and drive prior to debugging the design. The IP Catalog provides the VIO core under the Debug category. Detailed documentation on the VIO core can be found in the *LogiCORE IP Virtual Input/Output Product Guide (PG159)*.

This command returns a list of VIO debug core objects on the device, or returns an error if it fails.

Arguments

-of_objects *arg* - (Optional) Return the VIO debug cores of the specified hardware devices. The devices must be specified as objects using the **get_hw_devices** or the **current_hw_device** commands.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_hw_vios** based on property values on the debug cores. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "hw_vio" object, "NAME" and "INSTANCE_NAME" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 hw_vios against the specified patterns. The default pattern is the wildcard '*' which gets a list of all hw_vios available on the current hardware device.

Example

The following example gets the ILA debug cores defined on the current hardware device:

```
get_hw_vios -of_objects [current_hw_device]
```

See Also

- commit_hw_vio
- connect_hw_server
- current_hw_device
- get_hw_probes
- program_hw_devices
- refresh_hw_vio
- reset_hw_vio_activity
- reset_hw_vio_outputs
- set_property

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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-filter**]	Filter list with expression
[**-of_objects**]	Get interfaces of these pins or nets
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*patterns*]	Match IO standards against patterns Default: *

Categories

Object

Description

Get a list of IOSTANDARDs available for use 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_io_standards** 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 IOSTANDARDs object, "NAME", "IS_DCI" and "IS_DIFFERENTIAL" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 IOSTANDARDs of package_pin, site, bel, io_bank, and port objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 IOSTANDARDs against the specified patterns. The default pattern is the wildcard '*' which gets a list of all IOSTANDARDs for use on the target part. More than one pattern can be specified to find multiple IOSTANDARDs based on different search criteria.

NOTE: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets a list of differential IOSTANDARDs available for use on the target device:

```
get_io_standards -filter {IS_DIFFERENTIAL}
```

NOTE: If there are no objects matching the pattern you will get a warning.

See Also

- list_property
- report_property

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
[**-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 iobanks of these package_pins, ports or sites.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 "BANK_TYPE", "DCI_CASCADE", and "INTERNAL_VREF".

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
```

This example returns the GT Banks on the device:

```
get_iobanks -filter {BANK_TYPE == BT_MGT}
```

See Also

- get_package_pins
- get_ports
- get_sites
- list_property
- place_ports
- report_property

get_ip_upgrade_results

Get a list of results for IP upgrades during the current project.

Syntax

get_ip_upgrade_results [**-srcset** *arg*] [**-quiet**] [**-verbose**] [*objects*...]

Returns

List of IP upgrade results

Usage

Name	Description
[**-srcset**]	(Optional) Specifies the source file set containing the upgraded IP Default: The current source fileset Values: Source set name
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	IP to be upgraded Values: IP instance(s) within the design, as returned by 'get_ips ' or 'get_bd_cells <cell name>'

Categories

Object, Project, IPFlow

get_ipdefs

Get a list of IP from the current IP Catalog.

Syntax

```
get_ipdefs [-name] [-regexp] [-nocase] [-filter arg] [-quiet]
[-verbose] [patterns...]
```

Returns

List of Catalog IP objects

Usage

Name	Description
[**-name**]	Match the pattern against IP display name instead of VLNV
[**-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
[*patterns*]	The patterns to match against 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.

By default, the search is based on the VLNV property of the IP cores in the catalog. You can specify the **-name** option to search on the display name of IP cores instead.

Arguments

-name - (Optional) Indicates that the specified *pattern* refers to the DISPLAY_NAME property of the IP instead of the VLNV property.



TIP: *In the case of multi-word display names, such as "Multiply Adder", you can only search for a single string, so you would need to search for *Multiply* or *Adder* to locate this IP core*

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 [**-regexp**] [**-nocase**] [**-all**] [**-filter** *arg*] [**-quiet**] [**-verbose**] [*patterns...*]

Returns

List of IP objects

Usage

Name	Description
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching
[**-all**]	Return hidden IP
[**-filter**]	Filter list with expression
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match IP names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

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

-all - (Optional) Returns submodules of IP objects. Submodules are IP instances that are created as part of the generation of the parent IP core. By default, only parent IP objects used in the current project or design are returned, any IP cores used within those parent IP objects are not returned.

-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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. Valid objects are cells or instances (ie, get_cells), cell pins (ie, get_pins) and library pins (ie, get_lib_pins).
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
patterns	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 cells, pins, or library pins.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. Valid objects are cells or instances (ie, get_cells), cell pins (ie, get_pins) and library cells (ie get_lib_cells).
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
patterns	Match library cell pin names against patterns of the form /<library pin pattern>.

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

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 cells, library cells, or pin objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-filter**]	Filter list with expression
[**-of_objects**]	Get macros of these cells
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 of the specified cell objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_marked_objects

Get marked objects.

Syntax

get_marked_objects [**-rgb** *args*] [**-color** *arg*] [**-quiet**] [**-verbose**]

Returns

List of marked objects

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

Categories

Object, GUIControl

Description

Get the marked objects in the current design open in the Vivado IDE. Objects can be marked with the **mark_objects** command.

You can get all marked objects in the design, or specify marked objects by color, or by RGB value.

NOTE: This Tcl command works only when Vivado is run in GUI mode

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 used for marking the specified object or objects. Supported colors are: red, green, blue, magenta, yellow, cyan, and orange.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 all the marked objects in the current design:

```
get_marked_objects
```

The following example gets the object in the current design that are marked in the specified color:

```
get_marked_objects -color yellow
```

See Also

- get_highlighted_objects
- get_marked_objects
- highlight_objects
- mark_objects
- select_objects

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
[**-id**]	The message id to match. Should be used in conjunction with -limit or -count Default: empty
[**-severity**]	The message severity to match. Should be used in conjunction with -limit or -count Default: empty
[**-rules**]	Show a table displaying all message control rules for the current project
[**-limit**]	Show the limit for the number of messages matching either -id or -severity that will be displayed
[**-count**]	Show the number of messages matching either -id or -severity that have been displayed
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_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*] [**-to** *args*] [**-interconnect_only**] [**-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.
[**-to**]	Get the delay of the net to the given terminal(s) or port(s).
[**-interconnect_only**]	Include only interconnect delays. The default is to include the intra-site delay.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match net_delays against patterns Default: *

Categories

Timing, Netlist, Object

Description

Get delay objects for the specified nets in the current design, from the driver to each load pin, or to specified load pins, through specific pins.

The delay object includes properties containing the maximum and minimum delays for the fast and slow process corners. Use the **get_property** command to extract the property of interest from the delay object.

The values returned are calculated or estimated depending upon whether the net is routed. Delay values can include the actual delay of routed interconnect, or estimated net delays for unrouted nets. The net delay can also include delay through logic elements or device sites, or just include the interconnect delay.

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.

The **get_net_delays** command returns the delay objects for the specified nets, or returns an error if it fails.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_net_delays** based on property values on the delay objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the delay object, "NET", "FAST_MAX" and "FAST_MIN" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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* - (Required) Get the net delays of the specified nets. This option can be used to reduce the amount of data returned by the **get_net_delays** command.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

-to *args* - (Optional) Specifies the endpoints of nets for delay calculation. Pin, port, and pip objects can be specified as endpoints.

-interconnect_only - (Optional) Include only interconnect delays to determine the delay on the net due to routing. The default is to also include the intra-site delay into the delay calculation.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 net_delays against the specified patterns. The default pattern is the wildcard '*' which gets a list of all net_delays 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 the interconnect delay values for the specified net, and returns it in the form of a delay object:

```
report_property -all [lindex [get_net_delays -interconnect_only \
-of_objects [get_nets control_reg[*]]] 16 ]
```

See Also

- get_pins
- get_pips
- get_ports
- get_property
- list_property
- report_property

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, drc violations
[**-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 starting at the current instance. Without this argument, the command will only get nets from the current instance, as set by the **current_instance** command. 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 net name. For instance, searching for U1/* searches each level of the hierarchy for nets 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 cells, pins, ports, or clocks; or nets associated with specified DRC violation objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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.

-top_net_of_hierarchical_group - (Optional) Returns the top-level net segment of a hierarchical net, or nets, or the top-level net segments of all nets. Use this option with **-segment** to return the top-level net segment from all the segments of a hierarchical net.

-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 [upper | lower | both] - (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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 runs the **report_drc** command on the current design, returns the list of violations in the specified DRC report, and then returns any nets associated with the driverless net rule (NDRV):

```
report_drc -name drc_1
get_drc_violations -name drc_1
get_nets -of_objects [get_drc_violations -name drc_1 NDRV*]
```

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}
```

This example returns the net attached to the specified hierarchical pin object, then returns the net segments attached to the pin object, then returns the top-level net segment attached to the pin object:

```
get_nets -of [get_pins cpuEngine/or1200_cpu/or1200_sprs/esr_reg[9]_i_3/I0]
cpuEngine/or1200_cpu/or1200_sprs/flagforw
get_nets -segments -of [get_pins cpuEngine/or1200_cpu/or1200_sprs/esr_reg[9]_i_3/I0]
cpuEngine/or1200_cpu/or1200_alu/flagforw cpuEngine/or1200_cpu/flagforw
cpuEngine/or1200_cpu/or1200_sprs/flagforw
get_nets -top -segments -of \
[get_pins cpuEngine/or1200_cpu/or1200_sprs/esr_reg[9]_i_3/I0]
cpuEngine/or1200_cpu/flagforw
```

See Also

- connect_debug_port
- get_cells
- get_clocks
- get_drc_violations
- get_pins
- get_ports
- list_property
- report_drc
- 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, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 nets, tiles, nodes, bel_pins, site_pins, wires, or pip objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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*] [**-r**] [**-regexp**] [**-nocase**] [**-quiet**] [**-verbose**] [*patterns...*]

Returns

Returns all the objects found given the specified pattern

Usage

Name	Description
[**-filter**]	filters *patterns* according to the specified property-matching expressions
[**-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
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_objects** 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 HDL object, "NAME", "SCOPE" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 -recursive -filter {type == internal_signal} cpuEngine/cl*
   /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 -recursive -filter {type == internal_signal} ma* cl*
```

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, bels, iobanks, pkgpin_bytegroups, pkgpin_nibbles, or ports.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, bels, iobanks, pkgpin_bytegroups, pkgpin_nibbles, or ports.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 the package pins associated with the specified bytegroup of an UltraScale device:

```
get_package_pins -of [get_pkgpin_bytegroups BANK44_BYTE0]
```

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_package_pins
- get_pkgpin_bytegroups
- get_ports
- get_sites
- list_property
- place_ports
- 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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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*] [**-of_objects** *args*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

List of part 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 parts of the objects specified: project, design, or run.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match part names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 parts of the specified project, design, or run objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 clocks 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-filter**]	Filter list with expression
[**-of_objects**]	Get Pblocks of these cells
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, clocks, drc violations
[**-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

-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/TclCmd/regexp.htm.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** 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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

-of_objects *arg* - (Optional) Get the pins connected to the specified cell, clock, timing path, or net; or pins associated with specified DRC violation objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

This example runs the **report_drc** command on the current design, and then returns any pins associated with DRC violations:

```
report_drc -name drc_1
get_pins -of_objects [get_drc_violations]
```

See Also

- get_drc_violations
- list_property
- report_drc
- 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, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 sites, tiles, wires, nodes, pips, or nets objects. 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.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_pkgpin_bytegroups

Get a list of package pin byte groups.

Syntax

get_pkgpin_bytegroups [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-of_objects** *args*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Pin_group

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 pin_group of these package_pins, iobank, site, or port.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match pin_group against patterns Default: *

Categories

XDC, Object

Description

Gets a list of the byte groups on the I/O banks of the current Xilinx UltraScale device.

For 7 series devices, the hierarchy of IO Banks is divided into two object types: I/O Banks and Package Pins. For Xilinx UltraScale FPGA devices, the IO Bank hierarchy includes two additional divisions: Byte groups and Nibbles.

The relationships of these objects on an UltraScale device are defined as follows:

- An iobank has 2 or 4 bytegroups.
- Each pkgpin_bytegroup has 2 nibbles, an upper and lower, and has 13 package pins.
- Each pkgpin_nibble has 6 or 7 pins, and is the upper or lower nibble of the pkgpin_bytegroup.
- A package_pin is one pin of an iobank, a pkgpin_bytegroup, or a pkgpin_nibble.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_pkgpin_bytegroup** based on property values on the object. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the pkgpin_bytegroup object, "NAME", and "IOBANK" are two of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 pkgpin_bytegroups of the specified iobank, site, port, or package_pin objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 pkgpin_bytegroups against the specified patterns. The default pattern is the wildcard '*' which returns all pkgpin_bytegroups. 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_pkgpin_bytegroups -of [get_iobanks 44]
```

See Also

- get_iobanks
- get_package_pins
- list_property
- report_property

get_pkgpin_nibbles

Get a list of pkgpin nibbles.

Syntax

get_pkgpin_nibbles [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-of_objects** *args*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Pin_nibble

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 pin_nibble of these package_pins, iobank, site, or port.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match pin_nibble against patterns Default: *

Categories

XDC, Object

Description

Return a list of nibbles, or half-bytes, on the I/O banks of the current Xilinx UltraScale device.

For 7 series devices, the hierarchy of IO Banks is divided into two object types: I/O Banks and Package Pins. For Xilinx UltraScale FPGA devices, the IO Bank hierarchy includes two additional divisions: Byte groups and Nibbles.

The relationships of these objects on an UltraScale device are defined as follows:

- An iobank has 2 or 4 bytegroups.
- Each pkgpin_bytegroup has 2 nibbles, an upper and lower, and has 13 package pins.
- Each pkgpin_nibble has 6 or 7 pins, and is the upper or lower nibble of the pkgpin_bytegroup.
- A package_pin is one pin of an iobank, a pkgpin_bytegroup, or a pkgpin_nibble.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_pkgpin_nibbles** based on property values on the object. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the pkgpin_nibble object, "NAME", "IOBANK" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 pkgpin_nibble objects of the specified package_pins, iobank, site, or port.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 pkgpin_nibbles against the specified patterns. The default pattern is the wildcard '*' which returns all pkgpin_nibble objects on the current device.

Examples

The following example returns the Upper nibbles associated with the specified IO bank:

```
get_pkgpin_nibbles -of [get_iobanks 44] -filter {TYPE == U}
```

See Also

- get_iobanks
- get_package_pins
- 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
[**-regexp**]	Patterns are full regular expressions
[**-nocase**]	Perform case-insensitive matching (valid only when -regexp specified)
[**-filter**]	Filter list with expression
[**-of_objects**]	Get ports of these nets, instances, sites, clocks, timing paths, io standards, io banks, package pins, drc violations
[**-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 port names against patterns Default: *

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 nets, bels, sites, clocks, timing paths, io_standards, iobanks, or package_pins; or ports associated with specified DRC violation objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

The following example runs the **report_drc** command on the current design, returns the list of violations in the specified DRC report, and then returns the ports associated with any violations of the unspecified I/O Standard rule (NSTD):

```
report_drc -name drc_1
get_drc_violations -name drc_1
get_ports -of_objects [get_drc_violations -name drc_1 NSTD*]
```

See Also

- get_cells
- get_clocks
- get_drc_violations
- get_nets
- get_timing_paths
- list_property
- report_drc
- 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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` **`[-min]`** **`[-max]`** **`[-quiet]`** **`[-verbose]`** *`name object`*

Returns

Property value

Usage

Name	Description
[**-min**]	Return only the minimum value
[**-max**]	Return only the maximum value
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Name of property whose value is to be retrieved
object	Object to query for properties

Categories

Object, PropertyAndParameter, XDC

Description

Gets the current value of the named property from the specified object or objects. If multiple objects are specified, a list of values is returned.

If the property is not currently assigned to the object, or is assigned without a value, then the **get_property** command returns nothing, or the null string. If multiple objects are queried, the null string is added to the list of values returned.

If multiple objects are passed to the **get_property** command, you can use the -min or -max options to return the smallest or greatest value of the property specified by name. This feature can be useful when setting timing constraints.

RECOMMENDED: *For numeric properties, the min/max determination is based on numeric values. For all other properties, the determination is based on string sorting*

This command returns a value, or list of values, or returns an error if it fails.

Arguments

-min - (Optional) When multiple *objects* are specified, this option examines the values of the property specified by *name*, and returns the smallest value from the list of objects. Numeric properties are sorted by value. All other properties are sorted as strings.

-max - (Optional) When multiple *objects* are specified, this option examines the values of the property specified by *name*, and returns the largest value from the list of objects. Numeric properties are sorted by value. All other properties are sorted as strings.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

objects - (Required) One or more objects to examine for the specified property.

Examples

The following example gets the NAME property from the specified cell:

```
get_property NAME [lindex [get_cells] 3]
```

The following example returns the smallest PERIOD property from the specified clock objects:

```
get_property -min PERIOD [get_clocks]
```

This example demonstrates the string based sorting of the SITE property for the specified ports:

```
get_property SITE [get_ports]
IOB_X1Y75 IOB_X1Y76 IOB_X1Y98 IOB_X1Y125 IOB_X0Y94 IOB_X1Y95 IOB_X1Y96 IOB_X1Y93 IOB_X1Y94
get_property -min SITE [get_ports]
IOB_X0Y94
get_property -max SITE [get_ports]
IOB_X1Y98
```

NOTE: While IOB_X1Y125 is the largest site value on the port objects, the property value IOB_X1Y98 is returned because of the sorting of the property values as strings

See Also

- create_property
- get_cells
- get_ports
- 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
[**-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 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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**] [**-r**] [**-quiet**] [**-verbose**] [*patterns*...]

Returns

Returns HDL scope objects from the given arguments

Usage

Name	Description
[**-filter**]	filters *patterns* according to the specified property-matching expressions
[**-regexp**]	interprets *patterns* using regular expressions
[**-nocase**]	only when regexp is used, performs a case insensitive match
[**-r**]	only when a glob or regular expression pattern is used, descends recursively into children scopes to search for *patterns*
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	the pattern strings to search for scopes. Default: * (all children scopes)

Categories

Simulation

Description

Get a list of children HDL scopes of the current or specified scope

This command returns a list of scope objects, or returns an error.

Arguments

-recursive | **-r** - (Optional) Recursively return the children 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 returned results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_scopes** based on property values on the scope objects. You can find the properties on an object with the **report_property** or **list_property** commands.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Scope objects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all Scopes that are children of the current scope. More than one pattern can be specified to find multiple Scope objects based on different search criteria.

NOTE: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example recursively returns all of the children scopes of the specified scope:

```
get_scopes -r /testbench/dut
```

See Also

- current_scope
- report_scopes

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

- get_highlighted_objects
- get_marked_objects
- highlight_objects
- mark_objects
- 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, XDC

Description

Returns a list of site pins of the specified site, node, logical cell pin, or net objects in an open design.

This command recommends the use of the **-of_objects** argument to prevent high run times and compute resources.

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) Return the site pins of specified site, node, pin, or net objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 site pins against the specified patterns. The default pattern is the wildcard '*' which gets a list of all site pins of the specified objects. More than one search pattern can be specified to find site 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 site_pins of the specified Nets:

```
get_site_pins -of_objects [get_nets *Clk]
IOB_X1Y24/I
```

The following example returns the output site_pins associated with the site SLICE_X21Y92:

```
get_site_pins -of_objects [get_sites SLICE_X21Y92] -filter {DIRECTION==OUT}
SLICE_X21Y92/A SLICE_X21Y92/AMUX SLICE_X21Y92/AQ SLICE_X21Y92/B SLICE_X21Y92/BMUX
SLICE_X21Y92/BQ SLICE_X21Y92/C SLICE_X21Y92/CMUX SLICE_X21Y92/COUT SLICE_X21Y92/CQ
SLICE_X21Y92/D SLICE_X21Y92/DMUX SLICE_X21Y92/DQ
```

See Also

- get_nets
- get_nodes
- get_pins
- 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
[**-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 site_pips of these sites.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match site_pips against patterns Default: *

Categories

Object, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 PIPs of specified Sites.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 PIPs against the specified patterns. The default pattern is the wildcard '*' which gets a list of all PIPs for the Sites specified by **-of_objects**. 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_pips -of_objects [get_sites SLICE_X21Y92]
```

See Also

- get_sites
- list_property
- report_property

get_sites

Get a list of Sites.

Syntax

```
get_sites [-regexp] [-filter arg] [-nocase] [-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
[**-nocase**]	Perform case-insensitive matching. (valid only when -regexp specified)
[**-range**]	Match site names which fall into the range. Range is defined by exactly two site names.
[**-of_objects**]	Get the sites of slrs, tiles, bels, pins, package_pins, ports, pblocks, nets, site_types, io_banks, cells, clock_regions or drc_violation
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_USED", "NUM_INPUTS", and "NUM_OUTPUTS" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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; or sites associated with specified DRC violation objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 returns the number of sites that are not currently used on the device. Both command forms in the example return the same results. The second command directly evaluates the IS_USED boolean property:

```
llength [get_sites -filter {IS_USED==0}]
-or-
llength [get_sites -filter !IS_USED]
```

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_drc_violations
- get_property
- list_property
- report_property

get_slrs

Get a list of slrs.

Syntax

get_slrs [**-regexp**] [**-nocase**] [**-filter** *arg*] [**-of_objects** *args*] [**-quiet**] [**-verbose**] [*patterns*]

Returns

Slr

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 slr of these device, tiles, sites, bels, sitepins, belpins, clock region, node, pip, pin, package pin, iobank, cell.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match slr against patterns Default: *

Categories

Object, XDC

Description

Get a list of the super logic regions (SLRs) on the target device. On Devices that do not contain multiple SLRs, the SLR0 is 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_slrs** based on property values on the SLRs. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the SLR object, "NUM_CHANNELS", "NUM_SLLS" and "NUM_TILES" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 SLRs associated with the specified device, tiles, sites, bels, site_pins, bel_pins, clock_regions, nodes, pips, pins, package_pins, iobanks, or cells.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 SLRs against the specified patterns. The default pattern is the wildcard '*' which gets a list of all SLRs in the current design. More than one pattern can be specified to find multiple SLRs based on different search criteria.

NOTE: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example highlights each of the SLRs on the target device in a different color:

```
foreach x [get_slrs] {
   incr i
   highlight_objects -color_index $i $x
}
```

NOTE: If there are no cells matching the pattern you will get a warning.

See Also

- list_property
- report_property

get_speed_models

Get a list of speed_models in the device.

Syntax

get_speed_models [**-of_objects** *args*] [**-regexp**] [**-nocase**] [**-patterns** *arg*] [**-filter** *arg*] [**-quiet**] [**-verbose**]

Returns

Speed_models

Usage

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

Categories

Object, XDC

Description

Get speed models for the target part.

Speed files are provided by Xilinx for each device and speed grade. Speed files contain speed models. There are speed models for the various elements of a device: nodes, pips, bels. There are speed models for setup and hold, propagation delays, jitter, etc.

The speed models include information on the delays associated with device resources like BELs and SITEs and routing resources. Speed models are used by the Vivado timing engine to perform analysis of the current design in the context of the target part.

The objects returned are the speed models associated with specific physical resources like pips and wires, drawn directly from the speed files. These can include capacitance and resistance values and buffer models.

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.

This command returns the specified speed model objects, or returns an error if the command fails.

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_speed_models** 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 speed_model object, "NAME", "TYPE" and "DELAY" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 speed models for the specified node, bel, pip, or cell objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 on the speed model for FF type BELs:

```
report_property -all [lindex [ get_speed_models -of [get_bels *B6LUT]] 0]
get_speed_models: Time (s): cpu = 00:00:03 ; elapsed = 00:00:09 .
Memory (MB): peak = 2459.195 ; gain = 14.734
Property              Type    Read-only  Visible  Value
CLASS                 string  true       true     speed_model
DELAY                 double  true       true     0.043
FAST_MAX              double  true       true     0.035
FAST_MIN              double  true       true     0.028
IS_INSTANCE_SPECIFIC  bool    true       true     1
NAME                  string  true       true     bel_d_lut6_a1_o6
NAME_INTERNAL         string  true       false    bel_d_lut6_a1_o6:SLICEL:B6LUT
NAME_LOGICAL          string  true       true     bel_d_lut6_a1_o6
SLOW_MAX              double  true       true     0.043
SLOW_MIN              double  true       true     0.036
SPEED_INDEX           int     true       true     65535
TYPE                  string  true       true     bel_delay
```

See Also

- get_bels
- list_property
- report_property

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
[**-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 tiles of these slr, sites, bels, site_pins, bel_pins, nodes, wires, pips, nets, clock_regions.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	Match tiles against patterns Default: *

Categories

Object, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, bel_pins, nodes, wires, pips, nets, clock_regions, or slrs.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-from**]	List of pin or ports
[**-to**]	List of pin or ports
[**-filter**]	Filter list with expression
[**-of_objects**]	Get timing arcs for these cells
[**-quiet**]	Ignore command errors
[**-verbose**]	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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

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
[**-regexp**]	Patterns specified in filter are full regular expressions
[**-nocase**]	Perform case-insensitive matching for patterns specified in filter (valid only when -regexp 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.

-unique_pins - (Optional) Show only one timing path per each unique pin or group of pins. This is a boolean option, enabled by its use.

-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.

-user_ignored - (Optional) Show the timing paths that are ignored during timing analysis because the user has specified **set_false_path** or **set_clock_groups** timing constraints. This is a boolean option, enabled by its use.

-sort_by [slack | group] - (Optional) Sort timing paths by path groups, or by slack values. The default sort order is by slack values.

-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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

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}
```

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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[*]]\
-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 $path
```

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
```

The following example returns timing paths where the logic levels are greater than the specified number of logic levels:

```
get_timing_paths -max_paths 1000 -filter {LOGIC_LEVELS > 1}
```

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 hdl_objects. Allowed values are: default, dec, bin, oct,hex, unsigned, ascii
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hdl_object	The hdl_object 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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**]	interprets *patterns* using regular expressions
[**-nocase**]	only when regexp is used, performs a case insensitive match
[**-filter**]	filters *patterns* according to the specified property-matching expressions
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*patterns*]	the pattern strings to search for wave configuration names

Categories

Waveform

Description

Get the wave configuration objects that match the specified search options in the current simulation.

In the Vivado® simulator GUI, you can work with a waveform to analyze your design and debug your code. The Wave Config file contains the list of wave objects (signals, dividers, groups, virtual buses) to display, and their display properties, plus markers. A wave configuration displays with top-level HDL objects, and can be further populated using commands like **add_wave** and **add_wave_divider**.

This command returns the matching wave configuration objects, or returns nothing if no objects matched the 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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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_wave_configs** 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 "Wave Configuration" object, "NAME", "NEEDS_SAVE", and "FILE_PATH" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 wave configuration objects in the current simulation against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all open wave configurations in the simulation.

Examples

The following example returns all wave configuration objects, in the current simulation, that have unsaved changes:

```
get_wave_config -filter {NEEDS_SAVE}
```

See Also

- close_wave_config
- create_wave_config
- current_wave_config
- open_wave_config
- save_wave_config

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, XDC

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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*target_cell_name*]	Target cell
[*cells*]	Match engine names against cell names Default: *

Categories

IPIntegrator

Description

Create a new hierarchical module in the current IP Integrator subsystem design, and move the specified cells into that module.

You can also optionally move a group of specified cells into the hierarchical module. The connections between the specified cells are maintained. The connections between the cells being moved are maintained; connections between these cells and other cells that are not being moved are maintained automatically by IP Integrator adding pins and ports to cross the hierarchical boundary.

You can also move cells into the hierarchical module by using the **move_bd_cells** command after the hierarchical module has been created using the **create_bd_cells** command.

The command returns the name of the created hierarchical module if successful, or an error message if it fails.

Arguments

-prefix *arg* - (Optional) A prefix name to apply to any cells that are moved into the hierarchical module.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

target_cell_name - (Required) The name to assign to the hierarchical module that is created.

cells - (Required) The list of cells, specified by the **get_bd_cells** command, to move from the current IP subsystem design into the hierarchical module.

Example

The following example creates a hierarchical block in the current IP Integrator subsystem design, and moves the three specified cells into the block assigning them a name prefix as indicated:

```
group_bd_cells -prefix M1_ module1 [get_bd_cells /microblaze_1_xlconcat] \
[get_bd_cells /microblaze_1_axi_intc] [get_bd_cells /proc_sys_reset_1]
```

See Also

- get_bd_cells
- move_bd_cells

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-message 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
[**-message**]	Display information about the message with the given message. Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. Example: -message {Common 17-8}.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*pattern_or_object*]	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.

-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.

-syntax - (Optional) Returns only the syntax line for the command as a quick reminder of the proper form for using the command.

-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 the specified property of an object class, or the properties assigned to a specific object in the current design.

NOTE: This option requires the use of **-class**, or the specification of a single design object.

-class *arg* - (Optional) Return information related to the specified class of objects.

-message *arg* - (Optional) Return information related to the specified message. Messages are specified in the form of a unique global message ID, that consists of an application sub-system code and a message identifier: "Common 17-24", or {Common 17-24}. Refer to the **set_msg_config** command for more information.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_or_object - (Optional) Returns information related to the specified command, or a list of commands that match the specified pattern.

NOTE: A Vivado first class object, like a cell or site, must be specified when used with **-class** and **-prop** 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 -class cell -prop NAME
help -prop NAME [get_cells 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

- get_cells
- list_features
- list_property
- load_features
- report_property
- set_msg_config

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
[**-color_index**]	Color index
[**-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 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) Valid values are integers from 1 to 19. Specifies 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_highlighted_objects
- get_marked_objects
- get_selected_objects
- mark_objects
- select_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

Debug

Description

Implements the Vivado lab tools debug cores in the current project. 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_v3) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*files*]	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.



IMPORTANT: *Importing multiple files one at a time can cause noticeable performance degradation. It is more efficient to use a single* ***import_files*** *command to import a list of files:*

```
import_files {file1 file2 file3 ... fileN}
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 fileset.

```
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
[**-srcset**]	(Optional) Specifies the source file set containing the objects to be upgraded Default: The current source fileset Values: Source set name
[**-name**]	(Optional) Specifies a replacement name for the imported IP; may not be used with multiple files. Default: The current name for the imported IP Values: The new name for the imported IP
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*files*]	Names of the IP files to be imported Values: A list of XCI (and/or XCO) file name(s)

Categories

Project, IPFlow

Description

Adds an existing XCI or XCO file as an IP source into the current project, and copies it into the local project directory structure.

The **import_ip** command allows you to read existing IP files directly, and copy 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

-srcset *arg* - (Optional) Specifies the source file set to import the IP files into. If not specified, the default source file set is `sources_1`.

-name *arg* - (Optional) The name to assign to the IP object as it is added to the current source fileset. This option can only be used when a single file is specified in <*files*>.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) The names of the IP files to be imported into the current project. Each 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.

Examples

The following example copies 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
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-file_type**]	Input file type: 'csv' or 'xdc' Default: file type
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*file*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Issues an error that you can not run this command.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Issues an error that you can not run this command.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-of_objects args] [-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
[**-of_objects**]	Generate do file for this object (applicable with -scripts_only option only)
[**-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 QuestaSim, 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/QuestaSim 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.

-of_objects *arg* - (Optional) When used with **-scripts_only**, build the Modelsim launch script for a single specified subdesign, or composite file, to enable simulation of IP cores or other subdesigns.

NOTE: **-of_objects** can only be used with **-scripts_only**

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-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
[**-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
runs	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-lp 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
[**-lp**]	Add *repository_path* to the list of Driver/OS/Library search directories.
[**-hwspec**]	Specify the hardware platform specification file (*system*.xml)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

ToolLaunch

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.

-lp *arg* - (Optional) Specify the library, or repository path, for Driver/OS/Library search directories. This is a collection of libraries and drivers that form the lowest layer of your application software stack.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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/
```

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 netlist.

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, xsim, 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) Must be specified with **-mode** for post-synthesis or post-implementation simulations, but cannot be used with **-mode behavioral**. Specifies functional simulation of just the netlist, or timing simulation of the netlist and SDF file. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 functional simulation on a post implementation design:

```
launch_xsim -mode post-implementation -type functional
```

See Also

- close_sim
- current_sim
- xsim

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
filesize	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 netlist files and constraints with the target part to create the design. This command is intended for use with netlist source files, such as files generated by third party synthesis tools, or Vivado synthesis through the **synth_design** 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.
```

For project based designs with RTL source files, use **launch_runs** to launch synthesis or implementation, and then use the **open_run** command to open the design.

For non-project based designs, use the **open_checkpoint** command to open a checkpoint 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.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

- launch_runs
- open_checkpoint
- open_run
- synth_design

list_features

List available features.

Syntax

```
list_features [-quiet] [-verbose]
```

Returns

Nothing

Usage

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

Categories

Tools

Description

In order to reduce the memory footprint of the Vivado Design Suite, there are groups of Tcl commands called "features" which are unavailable for use until you run a command from that feature set, or unless you explicitly load the feature using the **load_features** command.

This command lists 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*object*]	Object to query for properties
[*pattern*]	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 or class of objects, but also reports the property type and property value.

Arguments

-class *arg* - (Optional) Return the properties of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

NOTE: **-class** cannot be used together with an *object*

-regexp - (Optional) Specifies that the search *pattern* is written as a regular expression.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

pattern - (Optional) Match the available properties on the *object* or **-class** against the specified search pattern. The *pattern* applies to the property name, and only properties matching the specified pattern will be reported. The default pattern is the wildcard '*' which returns a list of all properties on the specified object.

NOTE: The search pattern is case sensitive, and most properties are UPPER case.

Examples

The following example returns all properties of the specified CELL object:

```
list_property [get_cells cpuEngine]
```

The following example returns the properties matching the specified search pattern from the BEL class of objects:

```
list_property -class bel *NUM*
```

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
name	Name of property whose legal values is to be retrieved
[*object*]	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 property value for the specified class of objects.

-class *arg* - (Optional) Return the property values of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

NOTE: **-class** cannot be used together with an *object*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) A single object on which to report properties.

NOTE: If you specify multiple objects you will get an error.

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
files	Source file for which the targets needs to be listed

Categories

Project

Description

List the targets that are available for a specified IP core, DSP module, or IP Subsystem. The following file types are accepted: .xci, .xco, .mdl, .bd, .bxml.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- 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.

In order to reduce the memory footprint of the Vivado Design Suite, there are groups of Tcl commands called "features" which are unavailable for use until you run a command from that feature set, or unless you explicitly load the feature using the **load_features** command.

For example, the **load_features simulator** command loads the commands for the Vivado simulator, as does directly launching the Vivado 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 can load the feature into the application memory using the **load_features** command without actually running the feature of the tool.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. The 'lock/unlock' will only applied on physically placed cells and routed nets.

Syntax

lock_design [**-level** *arg*] [**-unlock**] [**-export**] [**-quiet**] [**-verbose**] [*cell*]

Returns

Nothing

Usage

Name	Description
[**-level**]	specify the locking and unlocking level; Valid values are logical, placement, and routing. Default: placement
[**-unlock**]	Unlock cells, if cells are not specified, whole design is unlocked; '-level' parameter must be specified for unlocking.
[**-export**]	mark that the constraints can be exported.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*cell*]	Lock cells, if cells are not specified, whole design is locked. Notice only placed cells and routed nets will be locked. Default: *

Categories

Project

Description

This command is used in the Hierarchical Design Flows for Design Preservation and Partial Reconfiguration. Refer to the *Vivado Design Suite User Guide: Hierarchical Design (UG905)* for more information on these design flows, and the use of this command.

The **lock_design** command is used to lock down the placement and/or routing of a design, or of the specified cell of a design. After reading in an Out-of-Context (OOC) design checkpoint using the **read_checkpoint** command, the preservation level for the module must be defined.

This command sets the IS_LOC_FIXED, IS_BEL_FIXED, and IS_ROUTE_FIXED properties of the specified logic.

Arguments

-level *arg* - (Optional) Specify the level of the cell or design to preserve in the current design. As a default, the placement data is preserved. Accepted values are:

- **logical** - Preserves the logical design. Any placement or routing information is still used, but can be changed if the tools can achieve better results.
- **placement** - Preserves the logical and placed design. Any routing information is still used, but can be changed if the tools can achieve better results. This is the default setting.
- **routing** - Preserves the logical, placed and routed design. Internal routes are preserved, but interface nets are not. In order to preserve routing, the CONTAIN_ROUTING property must have been used on the Pblock during the OOC implementation. This ensures that there will be no routing conflicts when the OOC implementation is reused.

-unlock - (Optional) Unlock cells. If cells are not specified, the whole design is unlocked. The -level parameter must be specified for unlocking.

-export - (Optional) Permit the export the placement and routing data as an XDC file. The constraints of a locked design or cell can be exported using the **write_xdc** command. By default, the constraints of a locked design or cell cannot be exported.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - (Optional) Lock the specified cells in the design, if cells are not specified, the whole design is locked. The default is to lock all cells in the design.



TIP: *Only placed cells and routed nets will be locked at the level specified*

Examples

The following example locks the placement and routing data for the specified cells of the current design:

```
lock_design -level routing [get_cells usbEngine*]
```

See Also

- read_checkpoint
- write_xdc

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
hdl_objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-level**]	Number of levels to log (for HDL scopes) Default: 0
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hdl_objects*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-recursive**]	Searches recursively for objects
[**-r**]	Searches recursively for objects
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hdl_objects	Which hdl_objects to trace

Categories

Simulation

Description

Log simulation activity for the specified HDL objects for viewing using Vivado simulator waveform viewer.

In the Vivado simulator, an HDL object is an entity that can hold a value, such as a wire, signal, or register.

Unlike **add_wave**, this command does not add the waveform object to waveform configuration. It simply enables logging of waveform activity to the Vivado simulator waveform database (WDB).

This command returns nothing.

Arguments

-recursive | **-r** - (Optional) Recursively log the waveform activity of the specified HDL objects, and the children of the specified HDL objects.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 the HDL objects to include in the Wave Config 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

The following example logs the waveform activities for the specified HDL objects.

```
log_wave -r [get_objects /testbench/dut/*]
```

See Also

get_objects

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
value	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
ports	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
files	Source file for which the wrapper needs to be generated

Categories

Project, SysGen

Description

Create a Verilog or VHDL wrapper for instantiating a sub-design into the project.

The **make_wrapper** command will create a wrapper for DSP modules from System Generator or MathWorks MatLab, Embedded Processor Designs from the IP Integrator feature of the Vivado Design Suite, or any IP Integrator block design.

You can generate a wrapper to make the sub-design the top-level of a stand-alone design, or for instantiating a sub-design into an existing design. You can also generate a wrapper for a simulation test bench of the sub-design.

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, or returns an error if it fails.

Arguments

-top - (Optional) Create a top-level Verilog or VHDL wrapper for the specified source. The wrapper instantiates the sub-design as the top-level of the design hierarchy.

-testbench - (Optional) Create a simulation test bench for the specified 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 destination fileset for importing the wrapper file into the project. By default, the wrapper will be imported into sources_1.

-import - (Optional) Import the wrapper file into the project, adding it to the appropriate fileset.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. The **make_wrapper** command only supports the `.mdl` file format from System Generator for DSP, the `.slx` format from MathWorks MATLAB, and the `.bd` file format from the IP Integrator feature of the Vivado Design Suite.

Examples

The following example creates the instantiation template to integrate the specified IP Integrator block design into the design hierarchy of the current project:

```
make_wrapper -inst_template -fileset [get_filesets sources_1] \
-files [get_files C:/Data/design_1/design_1.bd]
```

See Also

- add_files
- create_bd_design
- create_sysgen
- generate_target
- get_filesets
- list_targets

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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_highlighted_objects
- get_marked_objects
- get_selected_objects
- highlight_objects
- select_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
[*parent_cell*]	Parent cell
[*cells*]	Match engine names against cell names Default: *

Categories

IPIntegrator

Description

Move IP Integrator cells into the specified hierarchical module within the current subsystem design. The connections between the cells being moved are maintained; connections between these cells and other cells that are not being moved are maintained automatically by IP Integrator adding pins and ports to cross the hierarchical boundary.

Cells in the IP subsystem design can also be copied into a hierarchical module using **copy_bd_objs**, and can be grouped and added to a hierarchical module using **group_bd_objs**.

This command returns the name of the *parent_cell* module when successful, or returns an error message if it failed.

Arguments

-prefix *arg* - (Optional) A prefix name to apply to any cells that are moved into the hierarchical module.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

parent_cell - The name of the hierarchical module to move cells into.

cells - (Optional) The list of cells, specified by the **get_bd_cells** command, to move from the current IP subsystem design into the hierarchical module.

Example

The following example :

```
move_bd_cells -prefix mod1_ /myModule1 [get_bd_cells /myAxiFifo_1]
/myModule1
```

See Also

copy_bd_objs

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
[*files*]	Name of the files to be moved

Categories

Project, Simulation

Description

Moves files returned by the **get_files** command from one fileset to another while maintaining the properties on the files.

This command returns the list of files that were moved, or an error if the command fails.

Arguments

-fileset *arg* - (Optional) The destination fileset to which the specified source files should be moved. If no fileset is specified the files are moved to the sources_1 fileset by default. An error is returned if the specified fileset does not exist.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 objects returned by the get_files command.

NOTE: You must specify file objects returned by the **get_files** command, and not simply specify file names

Examples

The following example moves the file, `top_full.xdc`, to the constrs_2 fileset.

```
move_files -fileset constrs_2 [get_files top_full.xdc]
```

See Also

get_files

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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_checkpoint

Open a design checkpoint in a new project.

Syntax

open_checkpoint [**-part** *arg*] [**-quiet**] [**-verbose**] *file*

Returns

Nothing

Usage

Name	Description
[**-part**]	Override the checkpoint part. Note that this may cause errors if the checkpoint contains xdef.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Design checkpoint file

Categories

Project

Description

Open a design checkpoint file (DCP), create a new in-memory project and initialize a design immediately in the new project with the contents of the checkpoint. This command can be used to open a top-level design checkpoint, or the checkpoint created for an out-of-context module.

When opening a checkpoint, there is no need to create a project first. The **open_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.

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

-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.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 opens the specified checkpoint file, and specifies the target part for the design:

```
open_checkpoint C:/Data/state1/checkpoint.dcp -part xc7k325tffg900-2
```

See Also

- current_design
- current_project
- read_checkpoint
- write_checkpoint

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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 copies the IP customization and opens the example project for the specified IP core in a new location:

```
open_example_project -dir C:/Data/examples -force [get_ips blk_mem*]
```

See Also

- create_ip
- 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

Description

Open the Hardware Manager in the Vivado Design Suite in either the Vivado IDE or in Tcl or batch mode. Opening the Hardware Manager is the first step in programming and/or debugging your design in Xilinx FPGA hardware. For more information refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)*.

Hardware Manager is a feature of the Vivado Design Suite which lets you interact with FPGA devices on a board. The features of the Hardware Manager include:

- Logic Debug or Logic Analyzer- Debugging FPGA programmable logic designs.
- Programming/Configuration - Program FPGA devices using JTAG and configuring flash memory devices connected to FPGAs.
- In-system Serial I/O debug - Adjust SERDES receive/transmit settings and measure transmission bit error rates.
- System Monitor - Control on chip system monitor and read system monitor temperature and voltage values.

The Hardware Manager uses a number of first class objects, like hw_server, hw_target, hw_device, and hw_ila. Each of these objects is related to other objects, and has properties that can be set or read by the **set_property** and **get_property** commands to configure or control its function in the Hardware Manager. Refer to the *Vivado Design Suite Properties Reference Guide (UG912)* for more information on these objects, or type:

```
help -class object
```

The steps to connect to hardware and program the target FPGA device are:

1. Open the hardware manager in the IDE. (**open_hw**)



TIP: *This step can be skipped if you are running in batch or Tcl mode.*

2. Connect to a hardware server running either on the local machine, or on a remote network accessible host. (**connect_hw_server**)
3. Open a hardware target on the connected hardware server. (**open_hw_target**)
4. Identify the Xilinx FPGA device on the open hardware target. (**current_hw_device**, **get_hw_devices**)
5. Associate the bitstream data programming file (.bit), and probes file (.ltx) if one exists, with the appropriate FPGA device. (**set_property**)
6. Program or download the programming file into the hardware device. **(program_hw_device**, **refresh_hw_device)**.

Note that you can run the Hardware Manager from within the Vivado tool without having a project or design open. You can open the Hardware Manager, connect to the hardware server, and program the device on the target by providing a bitstream file, and probes file for debugging.

You can close the Hardware Manager using the **close_hw** command.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 opens the Hardware Manager in the Vivado Design Suite:

```
open_hw
```

See Also

- close_hw
- connect_hw_server
- current_hw_device
- current_hw_server
- get_hw_devices
- get_hw_targets
- open_hw_target
- refresh_hw_device
- set_property

open_hw_target

Open a connection to a hardware target on the hardware server.

Syntax

open_hw_target [**-jtag_mode** *arg*] [**-quiet**] [**-verbose**] [*hw_target*]

Returns

Nothing

Usage

Name	Description
[**-jtag_mode**]	Jtag mode to enable hw jtag for the target
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_target*]	hardware target Default: current hardware target

Categories

Hardware

Description

Opens a connection to the specified hardware target of the connected hardware servers, or opens the current hardware target.

The hardware target is a system board containing a JTAG chain of one or more Xilinx FPGA devices that you can program with a bitstream file, or use to debug your design. Connections between hardware targets on the system board and the Vivado Design Suite are managed by the **vcse_server** application. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a list of supported JTAG download cables and devices.

Each hardware target can have one or more Xilinx FPGA devices to program, or to use for debugging purposes. The current device is specified or returned by the **current_hw_device** command.

Use the **open_hw_target** command to open a connection to one of the available hardware targets. The open target is automatically defined as the current hardware target. Alternatively, you can define the current target with the **current_hw_target** command, and then open the current target. The Vivado Design Suite directs programming and debug commands to the open target through the hardware server connection.

An open connection to the hardware target can be closed using the **close_hw_target** command.

The **open_hw_target** command returns connection messages from the hardware server, or returns an error if it fails.

Arguments

-jtag_mode [on | off] - (Optional) Enables hardware JTAG for the target. This is a boolean property of the hw_target object that can also be enabled using values of "1" or "True".

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_target - (Optional) Specify the hw_target object to open a connection to for programming and debug. The hw_target must be specified as an object as returned by the **get_hw_targets** or **current_hw_target** commands. If no target is specified, the Vivado Design Suite will open a connection to the **current_hw_target**.

Example

The following example opens the hardware target returned by the **get_hw_targets** command:

```
open_hw_target [lindex [get_hw_targets] 0]
```

The following example shows the flow of opening the Hardware Manager, making a connection to the vcse_server application running on a remote host, setting the current hardware target, and opening that target, with JTAG mode enabled:

```
open_hw
connect_hw_server -host jupiter -port 60001
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/210203327463A]
open_hw_target -jtag_mode on
```

See Also

- close_hw_target
- connect_hw_server
- current_hw_device
- current_hw_target
- get_hw_devices
- get_hw_targets

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
[**-name**]	Design name
[**-part**]	Target part
[**-constrset**]	Constraint fileset to use
[**-quiet**]	Ignore command errors
[**-verbose**]	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_fileset** for that purpose.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-read_only**]	Open the project in read-only mode
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
run	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.

This command is intended to open a synthesized or implemented design that was created from design runs in Project Mode in the Vivado Design Suite.

Use the **open_checkpoint** command to open a Non-Project based checkpoint 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.

Arguments

-name - (Optional) The name of the design to open.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- link_design
- open_checkpoint
- read_checkpoint
- write_checkpoint

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
file_name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- start_vcd
- stop_vcd

open_wave_config

Open a wave config.

Syntax

`open_wave_config` `[`**`-quiet`**`]` `[`**`-verbose`**`]` `[`*`filename`*`]`

Returns

The wave config opened

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*filename*]	the name of a WCFG file from which to create a new wave configuration and corresponding wave window

Categories

Waveform

Description

Open the specified Wave Config file (`.wcfg`) in the current simulation.

Vivado simulator uses a simulation debug data model to allow users to debug HDL source files using source code stepping, breakpoints, conditions, and waveform viewing tools. The debug data model contains HDL object and scope names, and maps them to memory addresses to let you examine the changing values of signals, variables and constants during the simulation.

The waveform database is separate from the Wave Config file that stores the waveform activity for the simulation. The Wave Config file contains just the list of wave objects (signals, dividers, groups, virtual buses) to display, and their display properties, plus markers. The waveform database (WDB) contains the event data, values changing over time, for all traced signals, whether displayed or not.

A wave configuration object is created in the current simulation with the **create_wave_config** command. A Wave Config file is written to disk by the use of the **save_wave_config** command, and can be opened with the **open_wave_config** command.

The **open_wave_config** command opens a Wave Config file and maps it to the data source in the current simulation.



IMPORTANT: *Any HDL objects that are specified in the Wave Config file that are not found in the current simulation will be ignored.*

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

filename - (Optional) Specify the path and *filename*of the Wave Config file to open.

The *filename* should have a suffix of `.wcfg`, and the file suffix will be assigned automatically if no other suffix is supplied.

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 opens the specified Wave Config file:

```
open_wave_config testbench.wcfg
```

See Also

- close_wave_config
- create_wave_config
- save_wave_config

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
wdb	file name

Categories

Simulation

Description

The **open_wave_database** command opens an existing static simulator database file (WDB) for the current project. This simulation is a static simulation, not live, and can only be used to review prior results.

NOTE: Many of the commands for running and resetting the simulation are not available in a static simulation

Vivado simulator uses a simulation debug data model to allow users to debug HDL source files using source code stepping, breakpoints, conditions, and waveform viewing tools. The debug data model contains HDL object and scope names, and maps them to memory addresses to let you examine the changing values of signals, variables and constants during the simulation. When the simulation completes, the simulation is written to a static simulator database file (WDB).

HDL objects can be added to the simulation waveform database using the **log_wave** command which enables logging of waveform activity for the specified objects to the Vivado simulator waveform database (WDB).

The waveform database is separate from the Wave Config file that stores the waveform activity for the simulation. The Wave Config file contains just the list of wave objects (signals, dividers, groups, virtual buses) to display, and their display properties, plus markers. The waveform database (WDB) contains the event data, values changing over time, for all traced signals, whether displayed or not.

A Wave Config file is written to disk by the use of the **save_wave_config** command, and can be opened with the open_wave_config command.

Use the **open_wave_database** command with the **open_wave_config** command to open a previously completed simulation for review in the Vivado IDE.



TIP: *Objects that were logged in the simulation waveform database, with the **log_wave** command, can be added posthumously to the wave configuration in a static simulation using the **add_wave** command.*

Arguments

wbd - specifies the path and filename of the WDB file to open.

Examples

The following example opens a WDB file with the specified name, then opens an associated Wave Config file, and finally uses the **current_fileset** command to open the simulation database in the Vivado IDE:

```
open_wave_database {C:/Data/project_xsim/testbench_behav.wdb}
open_wave_config {C:/Data/project_xsim/testbench_behav.wcfg}
current_fileset
```

See Also

- create_wave_config
- current_fileset
- open_wave_config
- save_wave_config

opt_design

Optimize the current netlist. This will perform the retarget, propconst, sweep and bram_power_opt optimizations by default.

Syntax

```
opt_design [-retarget] [-propconst] [-sweep] [-bram_power_opt]
[-remap] [-resynth_area] [-resynth_seq_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
[**-resynth_seq_area**]	Resynthesis (with Sequential optimizations)
[**-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.

Run this command prior to implementation to optimize the design and simplify the netlist before placing and routing the design.

The **opt_design** command performs the following optimizations by default:

- Retarget
- Constant Propagation
- Sweep
- Block RAM Power optimizations

NOTE: Using command-line options for specific optimizations results in **opt_design** performing only the specified optimizations and disabling all others, even the ones that are usually performed by default.

To perform LUT Remapping, you must specify **-remap**.

To perform area-based re-synthesis, you must specify **-resynth_area**, or **-directive ExploreArea**.

To perform sequential area-based re-synthesis, you must specify **-resynth_seq_area**, or **-directive ExploreSequentialArea**.

Arguments

-retarget - (Optional) Retarget one type of block to another when retargetting 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.

NOTE: The **-retarget** argument is optional, as are the other optimizations. However this optimization is run by default unless explicitly overridden by another optimization

-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.

NOTE: Specific BRAM cells can be excluded from this optimization using the **set_power_opt** command

-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.

-resynth_seq_area - (Optional) Perform re-synthesis to reduce both combinational and sequential logic. Performs a superset of the optimization provided by **-resynth_area**.

-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.
- **ExploreSequentialArea** - Run multiple passes of optimization, with an emphasis on reducing registers and related combinational logic.
- **AddRemap** - Run the default optimization, and include LUT remapping to reduce logic levels.
- **NoBramPowerOpt** - Runs **opt_design** without the default BRAM power optimization.
- **RuntimeOptimized** - Run the fewest iterations, trading optimization results for faster runtime.
- **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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-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 four default optimizations: retarget, constant propagation, sweep, and BRAM power optimization. The command returns detailed results with the **-verbose** switch:

```
opt_design -verbose
```

This example excludes specific BRAM cells from power optimization using the **set_power_opt** command, and then runs **opt_design** with the four default optimizations:

```
set_power_opt -exclude_cells [get_cells -filter {PRIMITIVE_TYPE =~ BMEM.*.*} \
   -of_objects [get_pins -leaf -filter {DIRECTION == IN} \
   -of_objects [get_nets -of_objects [get_pins clock/bufgctrl_clk_mld/O]]]]
opt_design
```

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 and **-bram_power_opt** are implicitly 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
- set_power_opt
- synth_design

phys_opt_design

Optimize the current placed netlist.

Syntax

```
phys_opt_design [-fanout_opt] [-placement_opt] [-routing_opt]
[-rewire] [-critical_cell_opt] [-dsp_register_opt] [-bram_register_opt]
[-bram_enable_opt] [-shift_register_opt] [-hold_fix] [-retime]
[-force_replication_on_nets args] [-directive arg] [-critical_pin_opt]
[-clock_opt] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-fanout_opt**]	Do cell-duplication based optimization on high-fanout timing critical nets
[**-placement_opt**]	Do placement based optimization on timing critical nets
[**-routing_opt**]	Do routing based optimization on timing critical nets
[**-rewire**]	Do rewiring optimization
[**-critical_cell_opt**]	Do cell-duplication based optimization on timing critical nets
[**-dsp_register_opt**]	Do DSP register optimization
[**-bram_register_opt**]	Do BRAM register optimization
[**-bram_enable_opt**]	Do BRAM enable optimization
[**-shift_register_opt**]	Do Shift register optimization
[**-hold_fix**]	Attempt to improve slack of high hold violators
[**-retime**]	Do retiming optimization
[**-force_replication_on_nets**]	Force replication optimization on nets
[**-directive**]	Mode of behavior (directive) for this command. Please refer to Arguments section of this help for values for this option Default: Default
[**-critical_pin_opt**]	Do pin-swapping based optimization on timing critical nets
[**-clock_opt**]	Do clock skew optimization in post-route optimization
[**-quiet**]	Ignore command errors
[**-verbose**]	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 can be as post-place optimization and also as post-route optimization.



RECOMMENDED: *Optimization will not be performed on designs without negative slack*

Post-place **phys_opt_design** 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
- a final fanout optimization

NOTE: Using command-line options for specific optimizations results in **phys_opt_design** performing only the specified optimizations and disabling all others, even the ones that are usually performed by default.

Post-route **phys_opt_design** performs the following optimizations by default:

- placement-based optimization of critical paths
- routing optimization
- rewire
- critical-cell optimization

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 ExploreWithHoldFix** 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.

NOTE: The **-fanout_opt** argument is optional, as are the other optimizations. However this optimization is run by default unless explicitly overridden by another optimization

-placement_opt - (Optional) Move cells to reduce delay on timing-critical nets.

-routing_opt - (Optional) Perform routing optimization on timing-critical nets to reduce delay.

-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.

-bram_enable_opt - (Optional) The block RAM enable optimization improves timing on critical paths involving power-optimized block RAMs. Pre-placement block RAM power optimization restructures the logic driving block RAM read and write enable inputs to reduce dynamic power consumption. After placement the restructured logic may become timing-critical. The block RAM enable optimization reverses the enable-logic optimization to improve the slack on the critical enable-logic paths.

-shift_register_opt - (Optional) Perform shift register optimization to improve timing on negative slack paths between shift register cells (SRLs) and other logic cells. If there are timing violations to or from shift register cells (SRL16E or SRLC32E), the optimization extracts a register from the beginning or end of the SRL register chain and places it into the logic fabric to improve timing. The optimization shortens the wire length of the original critical path. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information.

-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.

-force_replication_on_nets *args* - (Optional) Force the drivers of the specified net objects to be replicated, regardless of timing slack. This option requires the use of the **get_nets** command to specify net objects. 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.

-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 replication for very high fanout nets.
- **ExploreWithHoldFix** - 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.
- **AlternateFlowWithRetiming** - Perform more aggressive replication and DSP and BRAM optimizations, and enable register re-timing.
- **AddRetime** - Performs the default **phys_opt_design** flow and adds 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**

-critical_pin_opt - For LUT inputs, this optimization performs remapping of logical pins to physical pins, also known as pin-swapping, to improve critical path timing. A critical path traversing a logical pin that has been mapped to a slow physical pin, such as A1 or A2, is reassigned to a faster physical pin, such as A6 or A5 if it improves timing.

NOTE: A cell with a LOCK_PINS property will be skipped and the cell will retain the pin mapping specified by LOCK_PINS. Logical-to-physical pin mapping can be obtained with **get_site_pins** for a selected logical pin object.

-clock_opt - (Optional) Perform clock skew optimization during post-route optimization. Insert global clock buffers to delay destination clocks to improve setup on critical paths.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 sets the LOCK_PINS property on the specified cell, then performs physical optimizations including register re-timing, optimization of registers across DSP blocks and block RAMs, and pin swapping (excluding the locked pins) to improve timing:

```
set_property LOCK_PINS {I3:A1 I2:A4} [get_cell cpuEngine/qmem_dack_reg_i_1]
phys_opt_design -retime -dsp_register_opt -bram_register_opt -critical_pin_opt
```

This example directs **phys_opt_design** to run more iterations, with hold violation fixing, to achieve potentially better results:

```
phys_opt_design -directive ExploreWithHoldFix
```

This example directs **phys_opt_design** to consider more nets for replication:

```
phys_opt_design -directive AggressiveFanoutOpt
```

See Also

- get_pins
- get_site_pins
- 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
cell_site_list	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 returns nothing if successful, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-timing_summary]
[-unplace] [-post_place_opt] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-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
[**-timing_summary**]	Enable accurate post-placement timing summary.
[**-unplace**]	Unplace all the instances which are not locked by Constraints.
[**-post_place_opt**]	Run only the post commit optimizer
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Tools

Description

Place the specified ports and logic cells in the current design, or all ports and logic cells, 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.
- **WLDrivenBlockPlacement** - 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.

-unplace - (Optional) Unplace all the instances which are not locked by constraints. Cells with fixed placement (IS_LOC_FIXED set to true), are not affected.

NOTE: Use the **set_property** to change IS_LOC_FIXED to False prior to unplacing fixed cells.

-post_place_opt - (Optional) Run optimization after placement to improve critical path timing at the expense of additional placement and routing runtime. This optimization can be run at any stage after placement, and can be particularly effective on a routed design. The optimization examines the worst case timing paths and tries to improve placement to reduce delay.

NOTE: Any placement changes will result in unrouted connections, so route_design will need to be run after -post_place_opt.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, runs optimization, routes the design, runs post placement optimization, and then reroutes the design to cleanup any unconnected nets as a result of post placement optimization:

```
place_design
phys_opt_design
route_design
place_design -post_place_opt
phys_opt_design
route_design
```

The following example directs the Vivado placer to try different placement algorithms to achieve a better placement result:

```
place_design -directive Explore
```

This example unplaces the current design:

```
place_design -unplace
```

See Also

- launch_runs
- opt_design
- place_ports
- phys_opt_design
- power_opt_design
- read_checkpoint
- route_design
- set_property

place_pblocks

Resize Pblocks according to SLICE demand and re-position them according to connectivity.

Syntax

```
place_pblocks [-effort arg] [-utilization arg] [-quiet]
[-verbose] pblocks...
```

Returns

Nothing

Usage

Name	Description
[**-effort**]	Placer effort level (per Pblock) Values: LOW, MEDIUM, HIGH Default: HIGH
[**-utilization**]	Placer utilization (per Pblock)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
pblocks	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) The percentage of slice resources on the target device that should be consumed by the logic elements assigned to a Pblock. For instance, a utilization rate of 50% means that half of the slice resources within the Pblock area should be allocated for use by the Pblock, and half should be left for use by other logic in the design. A high utilization value makes the Pblock smaller and more dense, but can make the overall design more difficult to place.

NOTE: Pblock utilization considers only slice-based device resources, and is estimated based on the post-synthesis logic assigned to a Pblock. Actual placement results may vary, and may require you to resize the Pblock using the **resize_pblock** command.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-skip_unconnected_ports**]	Do not place unconnected ports
[**-check_only**]	Only check IO/Clock placement DRCs
[**-iobank**]	Limit placement to the following banks
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*ports*]	Ports to place (if omitted, all ports will be placed). If the arguments are interleaved objects of ports and package pins, then manual placement is performed

Categories

PinPlanning

Description

Assign ports to the pins of the Xilinx FPGA package, by automatically or manually placing ports.

- Automatically places ports on an available I/O or clocking site, or into the specified I/O banks.
- Manually assigns ports to the specified package_pin when both the port and pin are specified.

The place_ports command will not replace ports that are currently placed by the user, or ports that are 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 as 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 unplaced ports will be placed.
- Ports can be specified in *port_name* and *package_pin* pairs to manually assign ports to the specified package_pin. Multiple port_name package_pin pairs can be specified as follows:

```
{port_name1 package_pin1 port_name2 package_pin2 port_name2 package_pin2 ...
port_nameN package_pinN}
```

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 follow example uses port_name package_pin pairs to manually place multiple ports:

```
place_ports {LEDS_n[2] AA11 LEDS_n[3] AA10 LEDS_n[0] Y11 LEDS_n[1] Y10}
```

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
[**-quiet**]	Ignore command errors
[**-verbose**]	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.

NOTE: Block RAM power optimizations are performed by default with the **opt_design** command. You can disable BRAM optimization by changing the defaults of opt_design, or by excluding specific cells from optimization using the **set_power_opt** command

You can also use the **read_saif** or **read_vcd** commands prior to optimization, and **power_opt_design** will consider the activity data while optimizing the design.

You can 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
```

This example optimizes the design, excluding the BRAM power optimization by specifying the optimizations to run, and then runs power optimization on the design:

```
opt_design -retarget -propconst -sweep
power_opt_design
```

See Also

- opt_design
- phys_opt_design
- read_saif
- report_power
- report_power_opt
- set_power_opt

pr_verify

Verify whether the design check points are replaceable on board. This command supports two formats:.

Syntax

pr_verify [**-full_check**] [**-file** *arg*] [**-initial** *arg*] [**-additional** *arg*] [**-quiet**] [**-verbose**] [*file1*] [*file2*]

Returns

Nothing

Usage

Name	Description
[**-full_check**]	Default behavior is to report the first difference only; if this option is set to true, pr_verify will report complete difference in placement or routing
[**-file**]	Filename to output results to. Send output to console if -file is not used.
[**-initial**]	Initial checkpoint (.dcp)
[**-additional**]	Additional checkpoints (.dcp)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*file1*]	Design checkpoint (.dcp) file one
[*file2*]	Design checkpoint (.dcp) file two

Categories

FileIO

Description

This command is used in the Partial Reconfiguration flow.

For Partial Reconfigurable designs to work in hardware, the placement and routing of static logic must be consistent between all configurations. In addition, proxy logic must be placed in the same locations and clock spine routing must match. The **pr_verify** command compares routed design checkpoint files (DCP) created for a Partial Reconfiguration design to verify that all imported resources match. For more information refer to the *Vivado Design Suite User Guide: Partial Reconfiguration (UG909)*.

This command returns the results of the comparison, or returns an error if it fails.

Arguments

-full_check - (Optional) Run a complete check of the two design checkpoints to report all mismatched resources. By default the **pr_verify** command will stop after the first mismatch since the two design checkpoints are not valid for partial reconfiguration.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

file1 - The first design checkpoint file to compare.

file2 - The second design checkpoint file to compare.

Examples

The following example verifies the two specified design checkpoint files for use in partial reconfiguration:

```
pr_verify -verbose C:/Data/FastConfig.dcp C:/Data/SlowConfig.dcp
```

See Also

- read_checkpoint
- write_checkpoint

program_hw_cfgmem

Program Cfgmem object.

Syntax

`program_hw_cfgmem` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`hw_cfgmem`*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_cfgmem	Cfgmem data to program

Categories

Hardware

Description

Erase, blank check, program, and/or verify the specified hw_cfgmem object with the memory configuration file defined in the object's PROGRAM.FILE property. The memory configuration file is created with the **write_cfgmem** command, and associated with the hw_cfgmem object using the set_property command as shown in the example.

The process whereby the design specific data is loaded or programmed into the Xilinx® FPGA is called configuration. The **create_hw_cfgmem** command defines a flash memory device used for configuring and booting the FPGA device.

After the hw_cfgmem object is created, and associated with the hw_device, the configuration memory can be programmed with the bitstream and other data from a memory configuration file created with the **write_cfgmem** command. The hw_cfgmem object is programmed using the **program_hw_cfgmem** command.

The **program_hw_config** command will run a multi-step process to erase the configuration memory device, perform a blank check to validate that the device is empty, program the device with the memory configuration file, and verify the programming on the device. Properties on the hw_cfgmem object determine which steps of the programming process are performed. These properties include:

- PROGRAM.FILE - Specifies the memory configuration file to use for programming the device. The memory configuration file is created with the **write_cfgmem** command.
- PROGRAM.ADDRESS_RANGE - Specifies the address range of the configuration memory device to program. The address range values can be:
  - **{use_file}** - Use only the address space required by the memory configuration file to erase, blank check, program, and verify.
  - **{entire_device}** - Erase, blank check, program, and verify the entire device.
- PROGRAM.ERASE - Erases the contents of the flash memory when true. This is a boolean property with a value of 0 (false) or 1 (true).
- PROGRAM.BLANK_CHECK - Checks the device to make sure the device is void of data prior to programming. This is a boolean property with a value of 0 (false) or 1 (true).
- PROGRAM.CFG_PROGRAM - Program the device with the specified PROGRAM.FILE. This is a boolean property with a value of 0 (false) or 1 (true).
- PROGRAM.VERIFY - Verify the device after programming. This is a boolean property with a value of 0 (false) or 1 (true).

This command returns a transcript of its process when successful, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_cfgmem - (Required) Specify a hw_cfgmem object to program. The hw_cfgmem must be specified as an object, rather than simply by name.

NOTE: You can use the **get_property PROGRAM.HW_CFGMEM** command on the **current_hw_device** to specify the hw_cfgmem object, or you can save the hw_cfgmem object to a Tcl variable when you create it with the **create_hw_cfgmem** command.

Example

The following example creates a hw_cfgmem object associated with the current_hw_device, defines the memory configuration file created from the bitstream with the write_cfgmem command, defines other properties of the programming process, and then programs the hw_cfgmem object:

```
set cfgMem [create_hw_cfgmem -hw_device [current_hw_device] [lindex $cfgParts 0 ]]
set_property PROGRAM.FILE {C:/Data/config_n25q128.mcs} $cfgMem
set_property PROGRAM.ADDRESS_RANGE  {use_file} $cfgMem
set_property PROGRAM.BLANK_CHECK  1 $cfgMem
set_property PROGRAM.ERASE  1 $cfgMem
set_property PROGRAM.CFG_PROGRAM  1 $cfgMem
set_property PROGRAM.VERIFY  1 $cfgMem
program_hw_cfgmem $cfgMem
```

NOTE: The hw_cfgmem object is stored in a Tcl variable, $cfgMem, so that it can be accessed later to set properties and program the object. The $cfgParts variable was defined in the example of the **get_cfgmem_parts** command

The following example programs the hw_cfgmem object using the **get_property** command to get the object as a property of the current hw_device:

```
program_hw_cfgmem [get_property PROGRAM.HW_CFGMEM [current_hw_device]]
```

See Also

- create_hw_cfgmem
- current_hw_device
- delete_hw_cfgmem
- get_cfgmem_parts
- get_property
- program_hw_cfgmem
- readback_hw_cfgmem
- set_property
- write_cfgmem

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

Description

Program the specified hardware device object or objects on the open hardware target of the current hardware server.

To access a Xilinx FPGA device through the Hardware Manager, you must use the following Tcl command sequence:

1. **open_hw** - Opens the Hardware Manager in the Vivado Design Suite.
2. **connect_hw_server** - Makes a connection to a local or remote Vivado CSE server application.
3. **current_hw_target** - Defines the hardware target of the connected server.
4. **open_hw_target** - Opens a connection to the hardware target.
5. **current_hw_device** - Specifies the Xilinx FPGA device to use for programming and debugging.

After connecting to the target hardware device, you must associate the bitstream file (.bit) from the design with the device, using the set_property command:

```
set_property PROGRAM.FILE {C:/Data/design.bit} [current_hw_device]
```

For debug purposes, you can also associate a probes file (.ltx) with the device using the PROBES.FILE property:

```
set_property PROBES.FILE {C:/Data/debug_nets.ltx} [current_hw_device]
```

Once the programming file has been associated with the hardware device, you can program the hardware device using the **program_hw_device** command, and debug the device using any of a number of Hardware Manager Tcl commands. To interactively debug the device open the Hardware Manager in the Vivado Design Suite IDE.

This command returns a transcript of its actions, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_device - (Optional) One or more hw_device objects to program. The hw_device must be specified as an object as returned by the **get_hw_devices** command. If the device is not specified, the **current_hw_device** will be programmed.

Example

The following example sets the current hw_device object, then sets the PROGRAM.FILE property for the device, and programs the device:

```
current_hw_device [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/Data/design.bit} [current_hw_device]
program_hw_devices [current_hw_device]
```

See Also

- current_hw_device
- connect_hw_server
- current_hw_target
- get_hw_devices
- get_hw_targets

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
value	value: on, true, yes. Otherwise set to off, false, no

Categories

Simulation

read_bd

Read one or more IPIntegrator design files.

Syntax

`read_bd [-quiet] [-verbose] files...`

Returns

List of IPIntegrator design file objects that were added

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
files	IPIntegrator design file name(s)

Categories

FileIO, IPIntegrator

read_checkpoint

Read a design checkpoint.

Syntax

```
read_checkpoint [-incremental] [-part arg] [-dont_reuse args]
[-only_reuse args] [-fix_reuse] [-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
[**-dont_reuse**]	Do not reuse given list of cells
[**-only_reuse**]	Reuse only given list of cells
[**-fix_reuse**]	Mark location of reused cells as fixed
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	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 design checkpoints at any stage in the design using the **write_checkpoint** command.

The **read_checkpoint** command simply reads the associated checkpoint file, without opening a design or project in-memory. To create a project from the imported checkpoint, use the **open_checkpoint** command instead of **read_checkpoint**, or use the **link_design** command after **read_checkpoint** to open the in-memory design from the checkpoint or checkpoint files currently read.

You can also import a design checkpoint, and re-target 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 placement and routing from the incremental DCP is applied to matching netlist objects in the current design to reuse existing placement and routing. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on the Incremental Compile flow.



RECOMMENDED: *The **-incremental** switch is not intended to merge two DCP files into a single design. It applies the placement and routing of the incremental checkpoint to the netlist objects in the current design*

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.

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 current design, using **open_run**, or read a new incremental checkpoint to overwrite the one previously loaded.

-dont_reuse *arg* - (Optional) When **-incremental** is specified, don't reuse the given list of cells from the incremental checkpoint.



TIP: ***-dont_reuse*** *and* ***-only_reuse*** *cannot be used together, but either can be used with* ***-fix_reuse***

-only_reuse *arg* - (Optional) When **-incremental** is specified, reuse only the specified list of cells from the incremental checkpoint.

-fix_reuse - (Optional) When **-incremental** is specified, mark the placement location of reused cells as fixed (IS_LOC_FIXED) to prevent changes by the **place_design** command. This is a boolean option that is true when specified.

-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.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 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 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
[**-quiet_diff_pairs**]	Suppress warnings about differential pair inference when importing I/O ports
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
files	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Name of the output file

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	hardware ILA data file name

Categories

Hardware

Description

Read ILA debug core data from the specified file, and create an hw_ila_data object.

The ILA debug sample data is acquired from a running device using the **upload_hw_ila_data** command. This creates a hw_ila_data object that can be written to a file on disk using the **write_hw_ila_data** command. This command reads that ILA data file.

The hw_ila_data object that is created by read_hw_ila_data is named after the *file* it is read from. If a hw_ila_data object of the same name already exists, the name of the object is assigned a number extension starting at 1: *file*_1.

The new hw_ila_data object is not connected with, or associated with, any ILA debug cores in the design.

After being read from disk, the ILA debug data can be viewed in the waveform viewer of the Vivado logic analyzer by using the **display_hw_ila_data** command.

This command returns an ILA data object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 ILA data file to read. If the file extension is not specified, the Vivado tool assumes an extension of `.ila`.

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.

Example

The following example reads the specified ILA data file, and creates a hw_ila_data object named after the file:

```
read_hw_ila_data C:/Data/hw_ila_data_2.ila
```

See Also

- current_hw_ila
- current_hw_ila_data
- display_hw_ila_data
- get_hw_ilas
- get_hw_ila_datas
- run_hw_ila
- write_hw_ila_data

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
file	hardware SIO scan file name
[*hw_sio_scan*]	hardware SIO scan data object Default: None

Categories

Hardware

Description

Read a hardware SIO scan data file and create a hw_sio_scan object in the Hardware Manager feature of the Vivado Design Suite.

The SIO scan data can be written to disk using the **write_hw_sio_scan** command, after running the scan using the **run_hw_sio_scan** command. This command reads that data file.

If no hw_sio_scan object is specified, a new hw_sio_scan object is created and is named sequentially following any existing hw_sio_scan objects. After being read from disk, the SIO scan data can be plotted and viewed in the Vivado serial I/O analyzer by using the **display_hw_sio_scan** command.

This command returns a hw_sio_scan object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 SIO scan data file to read. If the file extension is not specified, the Vivado tool assumes an extension of `.csv`.

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.

hw_sio_scan - (Optional) Specify an existing hw_sio_scan object to read the scan data file into. The hw_sio_scan object can be specified by name, or as an object returned by the **get_hw_sio_scans** command.

Example

The following example reads the specified SIO scan data file into an existing hw_sio_scan object:

```
read_hw_sio_scan C:/Data/LoopBack1.csv SCAN_0
```

See Also

- create_hw_sio_scan
- current_hw_device
- display_hw_sio_scan
- get_hw_sio_scans
- remove_hw_sio_scan
- run_hw_sio_scan
- stop_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

read_hw_sio_sweep

Read hardware SIO sweep data from a directory. A hardware SIO sweep object will be created if not provided.

Syntax

`read_hw_sio_sweep` [**`-quiet`**] [**`-verbose`**] *`directory`* [*`hw_sio_sweep`*]

Returns

Hardware SIO sweep object

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
directory	hardware SIO sweep directory name
[*hw_sio_sweep*]	hardware SIO sweep data object Default: None

Categories

Hardware

Description

Read a hardware SIO sweep data directory and create a hw_sio_sweep object in the Hardware Manager feature of the Vivado Design Suite.

The SIO sweep data can be written to disk using the **write_hw_sio_sweep** command, after running the sweep using the **run_hw_sio_sweep** command. This command reads the sweep directory containing multiple SIO scan data files.

If no hw_sio_sweep object is specified, a new hw_sio_sweep object is created and is named sequentially following any existing hw_sio_sweep objects. After being read from disk, any of the SIO scans in the sweep can be plotted and viewed in the Vivado serial I/O analyzer by using the **display_hw_sio_scan** command.

This command returns a hw_sio_sweep object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

directory - (Required) The name of the dirctory containing the SIO sweep data to read.

hw_sio_sweep - (Optional) Specify an existing hw_sio_sweep object to read the scan data files of the sweep into. The hw_sio_sweep object can be specified by name, or as an object returned by the **get_hw_sio_sweeps** command.

Example

The following example reads the specified sweep directory and creates a new hw_sio_sweep object, then displays one of the hw_sio_scans from that sweep:

```
read_hw_sio_sweep C:/Data/SWEEP_1/
display_hw_sio_scan [get_hw_sio_scans {SCAN_86}]
```

See Also

- create_hw_sio_sweep
- current_hw_device
- display_hw_sio_scan
- get_hw_sio_sweeps
- remove_hw_sio_sweep
- run_hw_sio_sweep
- stop_hw_sio_sweep
- wait_on_hw_sio_sweep
- write_hw_sio_sweep

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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
files	IP file name(s)

Categories

FileIO, IPFlow

Description

Read the specified list of IP files (XCI) 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.

When using the **read_ip** command all output products associated with the IP core, including the design checkpoint file (DCP) will be read into the in-memory design.



TIP: *In the project-based design flow, the Vivado tool will automatically generate the necessary output products associated with an IP core. However, in a non-project flow you must generate the necessary output products using the **synth_ip** or **generate_target** commands. For more information on working with IP refer to the Vivado Design Suite User Guide: Designing with IP (UG896)*

Use the **import_ip** command to add the IP cores and import the files into the local project directory.

This command returns the list of files read.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-strip_path**]	Specifies the name of the instance of the current design as it appears in the SAIF file
[**-no_strip**]	Do not strip first two levels of hierarchy from SAIF file
[**-out_file**]	Specifies the name of the output file that contains nets that could not be matched
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	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, or power optimization by **power_opt_design**. The **read_saif** command will annotate the design nodes with activity from the SAIF file and estimate power appropriately.

Running **report_power** or **power_opt_design** after reading the SAIF file will use the activity rates from the specified file during optimization and analysis.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

- power_opt_design
- report_power

read_schematic

Import schematic.

Syntax

`read_schematic` `[`**`-name`** *`arg`*`]` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`file`*

Returns

Name of the file previously exported

Usage

Name	Description
[**-name**]	Schematic window title
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Input file

Categories

FileIO

Description

Import a native schematic file that was previously exported from the Vivado Design Suite using the **write_schematic** command.

Arguments

-name *arg* - (Optional) The name of the Schematic window to open when reading the schematic file.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Schematic 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.

Example

The following example reads the specified schematic file and opens a Schematic window called "Sheet_1" in the Vivado IDE:

```
read_schematic C:/Data/mySchematic.txt -name Sheet_1
```

See Also

write_schematic

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
name	Name for the set of results
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_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
[**-sv**]	Enable system verilog compilation
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
files	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 fileset:

```
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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
files	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 of more files.

Syntax

```
read_xdc [-cells args] [-ref arg] [-quiet_diff_pairs] [-quiet]
[-verbose] files
```

Returns

List of files

Usage

Name	Description
[**-cells**]	Import constraints for these cells
[**-ref**]	Import constraints for this ref
[**-quiet_diff_pairs**]	Suppress warnings about differential pair inference when importing I/O ports
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
files	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 filenames of the XDC 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 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
```

NOTE: Multiple cells must be enclosed in braces, {}, or quotes, "".

See Also

- add_files
- infer_diff_pairs
- write_xdc

readback_hw_cfgmem

Readback data from the hw_cfgmem object.

Syntax

```
readback_hw_cfgmem [-all] [-verify] [-offset arg] -file arg
[-datacount arg] -hw_cfgmem arg [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-all**]	specify readback of all memory locations
[**-verify**]	compare data readback against corresponding locations in MEM_DATA_FILE
[**-offset**]	memory offset value Default: 0x0
-file	File to write readback to
[**-datacount**]	number of data units to readback Default: 0x0
-hw_cfgmem	Cfgmem data to program
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Hardware

redo

Re-do previous command.

Syntax

```
redo [-list] [-quiet] [-verbose]
```

Returns

With -list, the list of redoable tasks

Usage

Name	Description
[**-list**]	Show a list of redoable tasks
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_axi

Refresh hardware AXI object status.

Syntax

`refresh_hw_axi` `[`**`-quiet`**`]` `[`**`-verbose`**`]` `[`*`hw_axis`*`...]`

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_axis*]	List of hardware AXI objects.

Categories

Hardware

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
[**-update_hw_probes**]	Update hardware probe information, read from probes file
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_device*]	hardware device Default: current hardware device

Categories

Hardware

Description

Refreshes the in-memory view of the device by scanning for debug and IBERT cores on the specified hw_device object, and also reads a probe file when directed.

The Hardware Manager in the Vivado Design Suite creates, deletes, or updates the hw_ila, hw_vio, hw_sio*, and hw_axi objects based on the core information found in the device, and also what is read from the probes file in the case of ILA and VIO debug cores.

Use the **refresh_hw_device** after the **program_hw_devices** to keep the in-memory hardware debug objects in sync with the state of the actual cores on the physical device.

Arguments

-update_hw_probes *arg* - (Optional) Update the probes file (.ltx) associated with the specified device by reading the specified probes file.

NOTE: The probes file is associated with a hw_device object through the use of the **set_property** command to define the PROBES.FILE property.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_device - (Optional) Specify the hw_device object to refresh. The hw_device must be specified as an object as returned by the **get_hw_devices** or the **current_hw_device** commands. If the hardware device is not specified, the **current_hw_device** will be returned.

Example

The following example .....:

```
refresh_hw_device [lindex [get_hw_devices] 0]
```

See Also

- current_hw_device
- get_hw_devices
- program_hw_devices

refresh_hw_mig

Refresh the status of the current hardware object. Inputs can be any mig, device, target, or server 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_mig [-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
hw_objects	hardware objects

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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_server*]	hardware server

Categories

Hardware

Description

Refresh or reopen the connection to the current or specified hardware server.

This command returns the connection messages from the hardware server, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_server - (Optional) The hw_server to refresh. If the server is not specified, the **current_hw_server** will be refreshed. The hardware server can be specified by name, or specified as a hw_server object returned by the **get_hw_servers** or **current_hw_server** commands.

Example

The following example refreshes the current_hw_server:

```
refresh_hw_server
```

The following example refreshes all connected hardware servers, printing the host name prior to refreshing th server:

```
foreach x [get_hw_server] {puts "Refreshing Host $x"; refresh_hw_server $x}
```

See Also

- connect_hw_server
- current_hw_server
- disconnect_hw_server
- get_hw_servers

refresh_hw_sio

Refresh the status of the specified hardware objects. Inputs can be any serial I/O (except scan and sweep), device, target, or server 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
hw_objects	hardware objects

Categories

Hardware

Description

Refreshes the in-memory view of all of the properties, or specified properties, of the specified hw_sio objects with values read from the actual object on the hardware device.

Specified objects can include any serial I/O object such as GTs, RXs, TXs, PLLs, or Commons, excluding hw_sio_scan and hw_sio_sweep objects. SIO objects also include device, target, or server hardware objects.

The **refresh_hw_sio** command reads the values of the specified objects on the hardware device, and applies the value to the associated property of the IBERT core in the Hardware Manager.

This command returns no feedback of its operation if successful, or returns an error if it fails.

Arguments

-regexp - (Optional) The list of properties to refresh is defined using regular expression.

-properties *args* - (Optional) Refresh the specified property or properties of the hw_sio objects. As a default behavior, if no properties are specified, all properties of the specified object or objects will be refreshed from the current values on the hardware device.



TIP: *If a non-existent property is specified, that property is ignored.*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_objects - (Required) Specify one or more hw_sio objects to refresh. The hw_sio objects must be specified as objects returned by on of the **get_hw_sio_*** commands.

Example

The following example refreshes all properties on the specified hw_sio_gt object:

```
refresh_hw_sio [get_hw_sio_gts *MGT_X0Y11]
```

The following example refreshes all of the properties on all of the hw_sio objects on the current hardware device:

```
refresh_hw_sio [current_hw_device]
```

See Also

- current_hw_device
- get_hw_devices
- get_hw_servers
- get_hw_sio_commons
- get_hw_sio_gts
- get_hw_sio_iberts
- get_hw_sio_plls
- get_hw_sio_txs
- get_hw_targets
- report_property

refresh_hw_sysmon

Refresh the status of the current hardware object. Inputs can be hw_server, hw_target, hw_device or hw_sysmon objects. 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_sysmon [-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
hw_objects	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

Description

Refresh the connection to the specified hardware target on the current hardware server, and reload the hw_target object in the Hardware Manager of the Vivado Design Suite. If no hw_target object is specified, the **current_hw_target** will be refreshed.

The hardware target is a system board containing a JTAG chain of one or more Xilinx FPGA devices that you can program with a bitstream file, or use to debug your design. Connections between hardware targets on the system board and the Vivado Design Suite are managed by the **vcse_server** application. Refer to *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for a list of supported JTAG download cables and devices.

Each hardware target can have one or more Xilinx FPGA devices to program, or to use for debugging purposes. The current device is specified or returned by the **current_hw_device** command. After specifying the current hardware target, you can open the connection through the hardware target, to the Xilinx FPGA device using the **open _hw_target** command.

refresh_hw_target scans the devices on the hardware target and creates, deletes, or updates the hw_device objects available through the target. Available devices are returned using the **get_hw_devices** command.

This command returns a transcript of the refresh process, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_target - (Optional) Specify the hw_target object to refresh the connection to. The hw_target must be specified as an object as returned by the **get_hw_targets** or **current_hw_target** commands. If no target is specified, the Vivado tool will refresh the connection to the **current_hw_target**.

Example

The following example refreshes the current hardware target:

```
refresh_hw_target
```

See Also

- connect_hw_server
- current_hw_device
- current_hw_target
- get_hw_devices
- get_hw_targets
- get_hw_servers
- open_hw_target

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]`** **`[-quiet]`** **`[-verbose]`** *`hw_vios`*`...`

Returns

Nothing

Usage

Name	Description
[**-update_output_values**]	Update hardware probe OUTPUT_VALUE property with values read from VIO core(s). Default is false.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_vios	List of hardware VIO objects.

Categories

Hardware

Description

Update the INPUT_VALUE and ACTIVITY_VALUE properties of the input probes of the specified VIO debug cores with values read from the hw_vio core on the hardware device.

The Virtual Input/Output (VIO) debug core can both monitor and drive internal signals on a programmed Xilinx FPGA device in real time. The VIO core uses hardware probes, hw_probe objects, to monitor and drive signals on the device. Input probes monitor signals as inputs to the VIO core. Output probes drive signals to specified values from the VIO core.

The **refresh_hw_vio** command reads the signal values at the input probes of the VIO debug core on the device, and applies the value to the INPUT_VALUE property of the hw_probe, and updates the ACTIVITY_VALUE property on the probe as well.

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

Arguments

-update_output_values - (Optional) Update hardware probe OUTPUT_VALUE property with values read from the signals at the specified VIO debug cores. This is a boolean argument enabled by its presence. This option has the affect of resetting the OUTPUT_VALUE on the probe to match the signal value on the hw_vio debug core.



TIP: *This is the reverse of the **commit_hw_vio** command, which applies the OUTPUT_VALUE on the probe to the hw_vio debug core on the device*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_vios - (Required) Specify one or more hw_vio objects to refresh. The hw_vio objects can either be specified as objects returned by the **get_hw_vios** command, or specified by name.

Example

The following example refreshes the specified hw_vio debug core, specified by name, including updating the OUTPUT_VALUE property on the hw_probes:

```
refresh_hw_vio -update_output_values true hw_vio_1
```

See Also

- commit_hw_vio
- connect_hw_server
- current_hw_device
- get_hw_probes
- get_hw_vios
- program_hw_devices
- reset_hw_vio_activity
- reset_hw_vio_outputs
- set_property

regenerate_bd_layout

Regenerate layout.

Syntax

```
regenerate_bd_layout [-hierarchy arg] [-routing] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-hierarchy**]	Hierarchy path to the window
[**-routing**]	Preserve placement of blocks and regenerate routing
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

IPIntegrator

Description

Regenerate the layout of the current IP Integrator subsystem design in the open canvas. This command updates and redraws the graphical elements of the subsystem design in the Vivado IDE.

Arguments

-hierarchy *arg* - (Optional) Specify a hierarchical module to regenerate. Use the **get_bd_cells** command to specify the hierarchical module as an object.

-routing - (Optional) Refresh the routing in the IP Integrator canvas, but do not refresh the placement of objects.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 refreshes the IP Integrator canvas in the Vivado IDE:

```
regenerate_bd_layout
```

The following example refreshes the specified hierarchical module in the current block design:

```
regenerate_bd_layout -hierarchy [get_bd_cell myHier1]
```

See Also

- current_bd_design
- open_bd_design
- start_gui

register_simulator

Register specified simulator with Vivado application.

Syntax

`register_simulator` `[`**`-description`** *`arg`*`]` `[`**`-bootstrap`** *`arg`*`]` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`name`*

Returns

Nothing

Usage

Name	Description
[**-description**]	Specify a short description of the simulator
[**-bootstrap**]	Specify the command to 'load' a Tcl app or source a Tcl script
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Specify the name of the simulator to register

Categories

ToolLaunch, Simulation

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
[*files*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*BreakPointObjsOrIds*]	A list of one or more breakpoint objects and/or breakpoint object ID's to be removed

Categories

Simulation

Description

Remove specified breakpoints from the current simulation. You must have an open simulation to use this command.

A breakpoint is a user-determined stopping point in the source code used for debugging the design. When simulating a design with breakpoints, simulation of the design stops at each breakpoint to let you examine values and verify the design behavior.

The breakpoints in the current simulation can be reported using the **report_bps** command.

This command returns nothing, or an error if the command fails.

Arguments

-all - (Optional) Remove all breakpoints in the current simulation.



IMPORTANT: *This option will remove ALL breakpoints without warning, even if other options are specified*

-file *arg* - (Optional) Remove an existing breakpoint in the specified HDL source file.

-line *arg* - (Optional) Remove an existing breakpoint at the specified line number of the HDL source file.

NOTE: Both **-file** and **-line** must be used together to define an existing breakpoint

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

BreakPointObjsOrIds - (Optional) Specifies one or more breakpoint objects in the current simulation to remove. The breakpoint object is returned by the **add_bp** command when the breakpoint is added to the simulation.

Examples

The following example removes all the breakpoints in the current simulation:

```
remove_bps -all
```

See Also

report_bps

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
cells	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 the elaborated RTL design.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
pblock	Pblock to remove cells from
cells	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*ConditionObjs*]	ConditionObjs, id's or names

Categories

Simulation

Description

Remove specified conditions from the current simulation. You must have an open simulation to use this command.

Conditions can be defined prior to starting the simulation. When a condition is added, the simulator evaluates the condition expression anytime a signal change is detected. When a specified condition expression becomes TRUE, the condition commands are run.

The conditions in the current simulation can be reported using the **report_conditions** command.

This command returns nothing, or an error if the command fails.

Arguments

-all - (Optional) Remove all conditions in the current simulation.



IMPORTANT: *This option will remove ALL conditions without warning, even if other options are specified*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

ConditionObjs - Specifies one or more condition identifiers to remove from the current simulation. The condition identifiers are returned by the **add_condition** command when the condition is defined.

Examples

The following example removes the specified condition from the current simulation:

```
remove_conditions condition3
```

See Also

- add_condition
- report_conditions

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_check' 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_check' 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: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-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 to include a substring. See http://www.tcl.tk/man/tcl8.5/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/tcl8.5/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 filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

NOTE: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-ruledeck *arg* - (Required) The name of the rule deck to remove the specified design rule checks from.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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} -ruledeck 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 -ruledeck my_rules
```

See Also

- add_drc_checks
- create_drc_check
- create_drc_ruledeck
- get_drc_checks
- get_drc_ruledecks
- 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
files	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
```



CAUTION! *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
[*ForceObj*]	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_forces** 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
hw_sio_links	hardware SIO links

Categories

Hardware

Description

Removes the specified communication links between TX and RX objects on the GTs of the IBERT debug core defined on the current hardware device.

Vivado Serial I/O analyzer is a link-based analyzer, which lets you link between any transmitter and receiver within the IBERT design. The links define the communication paths and protocols between transmitters and receivers of the GigaBit transceivers on the device. This command removes those links.

This command returns a list of link objects on the IBERT debug core, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_links - (Required) Specify one or more hw_sio_link objects to remove. The hw_sio_link must be specified as an object as returned by the **create_hw_sio_link** or **get_hw_sio_links** commands.

Example

The following example removes the specified communication link on the IBERT debug core:

```
remove_hw_sio_link [get_hw_sio_links -filter {DESCRIPTION == "Link2"}]
```

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups

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
hw_sio_linkgroups	hardware SIO linkgroups

Categories

Hardware

Description

Removes the specified group that associates communication links between TX and RX objects on the GTs of the IBERT debug core defined on the current hardware device.

Vivado Serial I/O analyzer is a link-based analyzer. The links define the communication paths and protocols between transmitters and receivers of the GigaBit transceivers on the device. Link groups, or hw_sio_linkgroup objects, let you associate links into related groups, to collectively configure properties and run scans.



TIP: *The **remove_hw_sio_linkgroup** command removes the specified association, but does not remove the underlying communication links. Us the **remove_hw_sio_link** command to remove those objects*

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_linkgroups - (Required) Specify one or more hw_sio_linkgroup objects to remove. The hw_sio_linkgroup must be specified as an object as returned by the **create_hw_sio_linkgroup** or **get_hw_sio_linkgroups** commands.

Example

The following example removes the specified linkgroup:

```
remove_hw_sio_linkgroup [get_hw_sio_linkgroups {LINKGROUP_0}]
```

See Also

- create_hw_sio_link
- create_hw_sio_linkgroup
- current_hw_device
- get_hw_sio_iberts
- get_hw_sio_links
- get_hw_sio_linkgroups

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
hw_sio_scans	hardware SIO scans

Categories

Hardware

Description

Remove the specified serial I/O analyzer scan object.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_scans - (Required) Specify one or more hw_sio_scan objects to remove. The hw_sio_scan must be specified as an object as returned by the **create_hw_sio_scan** or **get_hw_sio_scans** commands.

Example

The following example removes the specified SIO scan:

```
remove_hw_sio_scan [get_hw_sio_scans {SCAN_2}]
```

See Also

- create_hw_sio_scan
- get_hw_sio_scans
- run_hw_sio_scan
- stop_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

remove_hw_sio_sweep

Remove an existing hardware SIO sweep.

Syntax

`remove_hw_sio_sweep` **`[-quiet]`** **`[-verbose]`** *`hw_sio_sweeps`*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_sio_sweeps	hardware SIO sweeps

Categories

Hardware

Description

Remove the specified serial I/O analyzer sweep scan object.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_sweeps - (Required) Specify one or more hw_sio_sweep objects to remove. The hw_sio_sweep must be specified as an object as returned by the **create_hw_sio_sweep** or **get_hw_sio_sweeps** commands.

Example

The following example removes the specified sweep scan object:

```
remove_hw_sio_sweep [get_hw_sio_sweeps {SWEEP_3}]
```

See Also

- create_hw_sio_sweep
- get_hw_sio_sweeps
- run_hw_sio_sweep
- stop_hw_sio_sweep
- wait_on_hw_sio_sweep
- write_hw_sio_sweep

remove_net

Remove nets from the current design.

Syntax

`remove_net [-prune] [-quiet] [-verbose]` *`nets...`*

Returns

Nothing

Usage

Name	Description
[**-prune**]	When performing net removal, remove pins and ports which are left unconnected as a result of the remove_net operation.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
nets	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. You can resize a bus, eliminating bits of the bus, using the **resize_net_bus** command.

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 the elaborated RTL design.

Arguments

-prune - (Optional) Prune, or remove, any unconnected hierarchical pins, ports, or nets, as a result of removing the specified net.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Using the following connection network:

```
leaf_cell1/pin1 > net1 > block1/pin1 > topnet < block2/pin1 < net2 < leaf_cell2/pin1
```

This example will remove block1/pin1, block2/pin1, net1, and net2, but will not prune the pins on the leaf cells:

```
remove_net topnet -prune
```

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: [Coretcl-82] No nets matched 'DataIn_pad_1_i[0]'.
remove_net DataIn_pad_1_i
```

See Also

- create_net
- disconnect_net
- resize_net_bus
- 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
pins	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.

To remove a bus pin, you must specify the primary pin name, and not specify a bus index. This ensures that the entire bus pin is removed, and not just a portion of the bits associated with the bus. You can resize a bus pin, eliminating bits, using the **resize_pin_bus** command.

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 the elaborated RTL design.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- create_net
- create_pin
- create_port
- resize_pin_bus
- 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
ports	Ports and/or bus ports to remove

Categories

PinPlanning

Description

Removes the specified ports or buses.

To remove a bus port, you must specify the primary port name, and not specify a bus index. This ensures that the entire bus port is removed, and not just a portion of the bits associated with the bus. You can resize a bus port, eliminating bits, using the **resize_port_bus** command.

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.

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 the elaborated RTL design.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_cell
- create_net
- create_pin
- create_port
- create_interface
- place_ports
- resize_port_bus
- write_checkpoint
- write_edif
- write_verilog
- write_vhdl

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
files	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 fileset:

```
reorder_files -front {C:/Data/FPGA/file1.vhdl C:/Data/FPGA/file2.vhdl}
```

NOTE: The default source fileset is used in the preceding example since the **-fileset** 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_fileset]
reorder_files -auto -disable_unused
```

See Also

- add_files
- create_fileset
- current_fileset
- remove_files

replace_bd_cell

Replace cell1 with cell2 by disconnecting connections to cell1 and connecting those connections to cell2.

Syntax

`replace_bd_cell` `[`**`-quiet`**`] [`**`-verbose`**`] [`*`cell1`*`] [`*`cell2...`*`]`

Returns

0 if success

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*cell1*]	Cell with connections that are to be disconnected.
[*cell2*]	Cell to be connected to connections that were disconnected from cell1.

Categories

IPIntegrator

Description

Move the connections currently assigned to one IP Integrator cell to another IP Integrator cell in the current design. This is intended to help you quickly replace on cell with another by moving connections from the source cell to the target cell.

Connections to the pins and interface pins on the cell are preserved where possible, and result in a Critical Warning when connections must be removed.

This command returns 0 if successful, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

cell1 - (Required) The IP Integrator cell to move connections from.

NOTE: The cell is not removed from the IP Integrator subsystem design.

cell2 - The IP Integrator cell to move connections to.

Example

The following example moves the connections from the specified cell, lmb_v10_1, to pins and interface pins of the same name on another cell, lmb_bram_cntlr_1:

```
replace_bd_cell [get_bd_cells /lmb_v10_1] [get_bd_cells /lmb_bram_if_cntlr_1]
CRITICAL WARNING: [BD 41-1164] The interface pin 'LMB_Sl_0' with bus definition
'xilinx.com:interface:lmb:1.0' is not found on the cell '/lmb_bram_if_cntlr_1'. Its
connection to the interface net 'Conn' has been removed.
CRITICAL WARNING: [BD 41-1166] The pin 'SYS_Rst' is not found on the cell
'/lmb_bram_if_cntlr_1'. Its connection to the net 'sys_rst_1' has been removed.
0
```

NOTE: Critical Warnings are returned when pin mismatches necessitate the remove of existing connections

See Also

- create_bd_cell
- get_bd_cells

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
[*BreakPointObjs*]	List of breakpoint objects to report

Categories

Simulation

Description

Report a specific breakpoint object, or report all breakpoints in the current simulation. You must have an open simulation for this command to return anything.

A breakpoint is a user-determined stopping point in the source code used for debugging the design. When simulating a design with breakpoints, simulation of the design stops at each breakpoint to let you examine values and verify the design behavior.

This command returns the filename and line number of the specified breakpoints, or of all breakpoints in the current simulation, or returns an error if the command fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

BreakPointObjs - Specifies one or more breakpoint objects in the current simulation to report. The breakpoint object is returned by the **add_bp** command when the breakpoint is added to the simulation.

Examples

The following example reports all breakpoints in the current simulation:

```
report_bps
```

This example reports the specified breakpoints in the current simulation:

```
report_bps bp1 bp2 bp5
```

See Also

- add_bp
- remove_bps

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 unclocked registers.

Syntax

```
report_clock_interaction [-delay_type arg] [-setup] [-hold]
[-significant_digits arg] [-no_header] [-file arg] [-append]
[-name arg] [-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-delay_type**]	Type of path delay: Values: max, min, min_max Default: max
[**-setup**]	Consider max delay timing paths (equivalent to -delay_type max)
[**-hold**]	Consider min delay timing paths (equivalent to -delay_type min)
[**-significant_digits**]	Number of digits to display: Range: 0 to 3 Default: 2
[**-no_header**]	do not generate a report header
[**-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, Timing

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.

-no_header - (Optional) Do not write a header to the report.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, Timing

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-file**]	Filename to output results to. (send output to console if -file is not used)
[**-append**]	Append to existing file
[**-write_xdc**]	file to output clock constraint. If not specified the clock constraint will be appended to clock report.
[**-return_string**]	return report as string
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-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
[*clocks*]	List of clocks Default: *

Categories

Report, Timing

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
[-sources] [-used_in arg] [-file arg] [-append] [-of_objects args]
[-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
[**-sources**]	Report the source compile order
[**-used_in**]	Specify the used in filter.
[**-file**]	Filename to output results to.
[**-append**]	Append output to existing file
[**-of_objects**]	Get 'file' objects of these types: 'file fileset ip'.
[**-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 or **-sources** option.

The -used_in option lets you report the processing order of files used in Synthesis, Simulation, or one of the implementation steps, according to the value of the USED_IN property.

By default the report is returned to the Tcl console, or standard output, but it can also be written to a file.

Arguments

-of_objects *args* - (Optional) Report the files that are associated with the specified filesets or sub-designs. The default is to search all filesets. When **-compile_order** and **-used_in** are specified, the **-of_objects** switch will only accept a single fileset, or a single sub-design such as an IP core, Block Design, or DSP design. A sub-design is also known as a composite file.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

-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.

-sources - (Optional) Report the compilation order of files found in the `sources_1` fileset of design sources.

-used_in *arg* - (Optional) Accepts one of the enumerated values of the **USED_IN** property for files, and returns files matching the specified value. Valid values for this option include the following: synthesis, simulation, or implementation. Refer to the *Vivado Design Suite Properties Reference Guide (UG912)* for information on the USED_IN property and its supported values.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- get_files
- 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
[*ConditionObjs*]	ConditionObjs, id's or names

Categories

Simulation

Description

Report a specific simulation condition, or report all conditions in the current simulation. You must have an open simulation for this command to return anything.

Conditions can be defined prior to starting the simulation. When a condition is added, the simulator evaluates the condition expression anytime a signal change is detected. When a specified condition expression becomes TRUE, the condition commands are run.

This command returns the condition identifier, expression, commands, and names of conditions, or returns an error if the command fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

ConditionObjs - (Optional) Specifies one or more condition identifiers in the current simulation to report. The condition identifiers are returned by the **add_condition** command when the condition is defined. BY default the command reports all conditions.

Examples

The following example reports conditions in the current simulation. The condition identifier, expression, commands, and names are reported:

```
report_conditions
#2: condition2
     Expression: {/testbench/reset == 0 }
     Command:    {
puts "Condition Reset was encountered at [current_time]. Stopping simulation."
stop }
     Name:       resetLow
#3: condition3
     Expression: {/testbench/leds_n == X000 }
     Command:    {
puts "Condition LED Unknown was encountered at [current_time]. Stopping simulation."
stop }
     Name:       ledUnknown
```

See Also

- add_condition
- remove_conditions

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
[**-file**]	Output the results to file
[**-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
[**-all**]	report all configuration settings (by default, only the typically important settings are reported
[**-no_header**]	do not generate a report header
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, Timing

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.

-name *arg* - (Optional) The name of the Datasheet Report view to display in the Vivado IDE when run in GUI mode. If the name has already been used in an open Report view, that view will be closed and a new report opened.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-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 a string
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Report, Debug

Description

Writes a report of the various Vivado device tool 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.

-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) Return report as a string. This argument can not be used with the **-file** argument.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-checks 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
[**-checks**]	DRC checks (see get_drc_checks for available checks)
[**-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, Timing

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 **report_drc** 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 are returned as Vivado objects that can be listed with the **get_drc_violations** command, and are associated with cells, pins, ports, nets, and sites in the current design. You can get the cells, nets, and other design objects that are associated with DRC violation objects, using the **-of_objects** option of the **get_cells** command for instance.

The Vivado tools include 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 **-checks** 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 **-checks**, 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.*

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.

-checks *args* - (Optional) A list of rule checks 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 **-checks** 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 **-checks** cannot be used together

-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.

-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

-format [text | xml] - (Optional) The default format of the output report is text. You can also output an XML report. XML output is only valid when **-file** is specified, and cannot be used with **-append**.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 -ruledeck placer_checks -file C:/Data/DRC_Rpt1.txt
report_drc -checks {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_ruledeck
- create_drc_violation
- get_cells
- get_drc_checks
- get_drc_ruledecks
- get_drc_violations
- get_nets
- get_pins
- get_ports
- get_sites
- 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
hdl_object	Which hdl_object to report

Categories

Simulation

Description

The **report_drivers** command prints the name and value of the driving signal, as well as the current value of a signal type HDL object.

Use this command to determine what signal or process is driving the value on a specific HDL signal, or net object. A driver of a signal is the statement in the HDL source file that is performing assignment to the signal.

The output format of **report_drivers** is as follows:

```
Drivers for <hdl_object>
  <Value of HDL Object>: Net <Hierarchical name of the probed signal>
    [ Declared Net : <The declared signal to which the probed signal is connected>]
    <Value of Driver> : Driver <Hierarchical name of the HDL process containing
the driver> at <file_name>:<line number>>
```

NOTE: The Declared Net is returned when the probed signal name is different from the hierarchical name of the actual declared signal due to the current scope of the simulation. Each bit of the declared net is printed for the probed signal

The values of signals returned by the **report_drivers** command depend on the state of the simulation. In the following example, the report is run before and after simulation:

```
current_scope /testbench/dut
report_drivers leds_n[3:0]
Drivers for /testbench/dut/LEDS_n[3:0]
  0 : Net /testbench/dut/LEDS_n[0]
    Declared Net : /testbench/leds_n[3]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    0 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    0 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    0 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
  0 : Net /testbench/dut/LEDS_n[1]
    Declared Net : /testbench/leds_n[2]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    0 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    0 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    0 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
  0 : Net /testbench/dut/LEDS_n[2]
    Declared Net : /testbench/leds_n[1]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    1 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    1 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    1 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
  X : Net /testbench/dut/LEDS_n[3]
    Declared Net : /testbench/leds_n[0]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    0 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    0 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    0 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
run all
report_drivers leds_n[3:0]
Drivers for /testbench/dut/LEDS_n[3:0]
  0 : Net /testbench/dut/LEDS_n[0]
    Declared Net : /testbench/leds_n[3]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    0 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    0 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    0 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
  1 : Net /testbench/dut/LEDS_n[1]
    Declared Net : /testbench/leds_n[2]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    0 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    0 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    0 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
  0 : Net /testbench/dut/LEDS_n[2]
    Declared Net : /testbench/leds_n[1]
    1 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    1 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    1 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    1 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
  0 : Net /testbench/dut/LEDS_n[3]
    Declared Net : /testbench/leds_n[0]
    0 : Driver /testbench/dut/line__187 at C:/Data/sources/sinegen_demo.vhd:187
    0 : Driver /testbench/dut/line__186 at C:/Data/sources/sinegen_demo.vhd:186
    1 : Driver /testbench/dut/line__185 at C:/Data/sources/sinegen_demo.vhd:185
    0 : Driver /testbench/dut/line__184 at C:/Data/sources/sinegen_demo.vhd:184
```

NOTE: Notice the declared net is reported, because the current scope of the simulation is set to a different level than the top-level of the test bench

This command returns a report of the drivers on the specified objects, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - Report the drivers of the specified VHDL signals or Verilog wires. The HDL objects can be specified by name, or returned by the **get_objects** command.

Examples

The following example reports the drivers for the HDL objects returned by the **get_objects** command:

```
report_drivers [get_objects leds_n]
```

See Also

- current_scope
- get_objects

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.

-format [text | xml] - (Optional) The default format of the output report is text. You can also output an XML report. XML output is only valid when **-file** is specified, and cannot be used with **-append**.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-summary]
[-coverage] [-no_header] [-file arg] [-append] [-return_string]
[-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
[**-ignored**]	only report exceptions which are ignored
[**-summary**]	report a summary of all exceptions
[**-coverage**]	report the coverage of all timing exceptions
[**-no_header**]	Do not generate a report header
[**-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

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.



IMPORTANT: *The **set_clock_groups** constraint is not reported by the **report_exception** command since it is not a considered a timing exception by Vivado timing analysis*

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 timing path exceptions in the current design that are ignored by the Vivado timing engine. Ignored constraints could be the result of an incorrectly defined constraint, or of missing design objects.

-summary - (Optional) Report a summary of all exceptions.

-coverage - (Optional) Report the coverage of all timing exceptions.

-no_header - (Optional) Do not write a header to the report.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
```

This example reports all timing exceptions ignored or overridden in the current design:

```
report_exceptions -ignored
```

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] [-timing]
[-histogram] [-load_types] [-clock_regions] [-slr] [-max_nets arg]
[-min_fanout arg] [-max_fanout arg] [-name 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
[**-timing**]	Report worst slack and worst delay values on nets
[**-histogram**]	Report histogram for high fanout nets
[**-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
[**-name**]	Output the results to GUI panel with this name
[**-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.

-timing - (Optional) Add timing date to display the worst slack (WNS) and worst delay on high fanout nets.

NOTE: The **-timing** option will slow the run time of **report_high_fanout_nets**, and is not supported with **-histogram**

-histogram - (Optional) Format the report as a histogram showing the number of fanouts, and the number and percentage of nets with that many fanouts in the design.

-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 that 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. The default value is 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.

-name *arg* - (Optional) The name of the High Fanout Nets Report view to display in the Vivado IDE when run in GUI mode. If the name has already been used in an open Report view, that view will be closed and a new report opened.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_hw_axi_txn

Report formatted hardware AXI Transaction data.

Syntax

report_hw_axi_txn [**-w** *arg*] [**-t** *arg*] [**-quiet**] [**-verbose**] *hw_axi_txns* ...

Returns

Nothing

Usage

Name	Description
[**-w**]	Output data bytes per output line. Default: 8
[**-t**]	d[SIZE] signed decimal, SIZE bytes per integer, b[SIZE] binary, SIZE bytes per integer, o[SIZE] octal, SIZE bytes per integer, u[SIZE] unsigned decimal, SIZE bytes per integer, x[SIZE] hexadecimal, SIZE bytes per integer Default: x4 (4-bytes in hex)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_axi_txns	hardware AXI Transaction object to report

Categories

Hardware

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 hierachical 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Description

This command examines the IP cores in the current project, and reports the state of the IP with regard to the latest IP catalog. The following information is included in the IP Status report:

- Instance Name - The name of the IP core in the current project.
- IP Status - A description of the state of the IP in the current project.
- Recommendation - A recommended action based on the status.
- Lock Status - An explanation of the lock status of the IP in the current project.
- Change Log - A reference to the change log for the IP update in the catalog. This will provide a description of the changes in the latest IP.
- IP Name - The name of the IP core in the catalog.
- IP Version - The version of the IP in use in the current project.
- New Version - The latest version of the IP in the catalog.

- New license - Any license changes required by the IP.
- Original Part - The original part associated with the IP in the catalog.

IP cores that are out of date, or locked, may need to be upgraded and the output products regenerated. Refer to the *Vivado Design Suite User Guide: Designing with IP (UG896)* for more information.

This command returns the IP status report, or returns an error if it fails.

Arguments

-name *arg* - (Optional) The name to assign to the results when run in GUI mode.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 IP status to the specified file, appending the results if the file already exists:

```
report_ip_status -file C:/Data/reports/ip_status.txt -append
```

See Also

- get_ips
- import_ip
- read_ip
- upgrade_ip

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

Description

The **report_objects** command reports the type, name, and language of the specified HDL objects to the Tcl Console or Tcl shell. You must have an open simulation to use this command.

This command returns a brief description of the specified objects. Use the **describe** command to return more detailed information.

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 command returns the HDL object type, the name, and the code type (Verilog/VHDL) for each object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 objects to report. The default command reports all objects found in the current scope of the simulation, or **current_scope**.

NOTE: Objects can be specified by name, or returned as objects by the **get_objects** command.

Examples

The following example shows how the specified objects reported depend upon the current scope of the simulation:

```
current_scope testbench
/testbench
report_objects [get_objects leds_n]
  Declared: {leds_n[3:0]}                 Verilog
current_scope dut
/testbench/dut
report_objects leds_n
       Out: {LEDS_n[3:0]}                 VHDL
```

This example reports the specified HDL objects of the current simulation scope:

```
report_objects [get_objects GPIO*]
```

See Also

- current_scope
- describe
- get_objects

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-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
[**-non_default**]	Report only params that are set to a non default value
[**-return_string**]	Return report as string
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*pattern*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-file**]	Output file
[**-append**]	Append the results to file
[**-return_string**]	return report as string
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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: power
[**-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_switching_activity** command.

Switching activity can also be read in from an SAIF file with the **read_saif** command, or from a VCD file using **read_vcd**. The Vivado tool will annotate the design nodes with activity from the SAIF or VCD file and estimate power appropriately.

Power analysis requires an open synthesized design, 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

-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 [power | logic | all] - (Optional) Displays the summary power consumption for each level of design hierarchy (**power**), or the power broken-down for different logic elements of the hierarchy (**logic**), or both the power summary and the different logic elements of the design hierarchy (**all**). The default is **power**.

-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

-format [text | xml] - (Optional) The default format of the output report is text. You can also output an XML report. XML output is only valid when **-file** is specified, and cannot be used with **-append**.

-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.

-fileset arg - (Optional) Fileset to search for the .xpe file for PS7 block.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- set_switching_activity
- set_operating_conditions

report_power_opt

Report power optimizations.

Syntax

```
report_power_opt [-cell args] [-file arg] [-format arg] [-name arg]
[-append] [-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-cell**]	list of instance names Default: empty
[**-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 *args* - (Optional) Report power optimization for the specified cell or cell instances. By default, the power optimizations performed on the whole design are reported.

-file *arg* - (Optional) Write the report into a 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.

-format [text | xml] - (Optional) Specify the file format of the Power Optimization report when used with **-file**. The default file output is a text format file. The Vivado tool can also write an XML format file to allow for use by third party applications.

NOTE: When **-format xml** is specified, the **-append** option is not supported

-name *arg* - (Optional) The name to assign to the Power Optimization report when the command is run in GUI mode.

-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, and is not compatible with the XML format

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, writing them to the specified file in an XML format:

```
report_power_opt -format xml -file C:/Data/power_opt.xml
```

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
[**-all**]	Report all properties of object even if not set
[**-class**]	Object type to query for properties. Not valid with *object*
[**-return_string**]	Set the result of running report_property in the Tcl interpreter's result variable
[**-file**]	Filename to output result to. Send output to console if -file is not used
[**-append**]	Append the results to file, don't overwrite the results file
[**-regexp**]	Pattern is treated as a regular expression
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*object*]	Object to query for properties
[*pattern*]	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, or class of objects.

NOTE: **list_property** also returns a list of all properties on an object, but does not include the property type or value.

You can specify objects for **report_property** using the **get_*** series of commands to get a specific object. You can use the **lindex** command to return a specific object from a list of objects:

```
report_property [lindex [get_cells] 0]
```

However, if you are looking for the properties on a class of objects, you should use the **-class**option instead of an actual object.

This command returns a report of properties on the object, or returns an error if it fails.

Arguments

-all - (Optional) Return all of the properties for an object, even if the property value is not currently defined.

-class *arg* - (Optional) Return the properties of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

NOTE: **-class** cannot be used together with an *object*

-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

-regexp - (Optional) Specifies that the search *pattern* is written as a regular expression.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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.

pattern - (Optional) Match the available properties on the *object* or **-class** against the specified search pattern. The *pattern* applies to the property name, and only properties matching the specified pattern will be reported. The default pattern is the wildcard '*' which returns a list of all properties on the specified object.

NOTE: The search pattern is case sensitive, and most properties are UPPER case.

Examples

The following example returns all properties of the specified object:

```
report_property -all [get_cells cpuEngine]
```

The following example returns the properties of the specified class of objects, rather than an actual object:

```
report_property -class bel
```

The following example returns properties on the **current hw_device** that match the specified pattern, specified as a regular expression:

```
report_property [current_hw_device] -regexp .*PROG.*
```

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
- current_design
- current_fileset
- current_hw_device
- current_project
- current_run
- 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
[*objects*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_ram_utilization

Report configuration about RAMs in design.

Syntax

```
report_ram_utilization [-append] [-file arg] [-return_string]
[-cells args] [-detail] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
[**-append**]	Append to existing file
[**-file**]	Filename to output results to. (send output to console if -file is not used)
[**-return_string**]	return report as string
[**-cells**]	Limits the reporting to only those memory arrays that are contained within the specified cells.
[**-detail**]	Adds individual cell utilization data underneath the default RAM summary
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Report

Description

Report the utilization of RAM resources on the target part by the current synthesized or implemented design. The report is returned to the standard output, unless the **-file** or **-return_string** arguments are used.

The report includes the usage of different type of memory resources on the device, including BlockRAM and distributed RAM, and can optionally provide added configuration details.

Though RAM 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. This command will overwrite any files of the same name without warning.

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.

-cells *arg* - (Optional) Report the RAM resources utilized by one or more hierarchical cells in the current design. The cells must be specified as objects, using **get_cells**, rather than by name.

-detail - (Optional) Reports various configuration properties of the RAM, like size, write and read width and depth, address, and LOC.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 provides a detailed report of the RAM resources utilized by specified cells, and writes the results to the specified file:

```
report_ram_utilization -cells [get_cells U_*] -detail -file C:/Data/ram_util.txt
```

See Also

- all_rams
- get_cells
- report_utilization

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.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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*]
```

See Also

route_design

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

Description

Reports the names and types of HDL Scopes in the current scope of the current simulation, or of specified scopes.

An HDL Scope is a declarative region of an HDL file, where objects are declared. The following are examples of HDL Scopes in Verilog and VHDL:

- Verilog scopes: module, function, task, process, other begin-end blocks
- VHDL scopes: entity/architecture pair, block, function, procedure, process

You must have an open simulation to use this command.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_scopes - (Optional) Specifies the scopes upon which to report. The default is the current scope.

Examples

The following example reports the children scopes of `/tb/UUT`:

```
report_scopes [get_scopes /tb/UUT/* filter {type==module}
```

The following example reports the children scopes of the current scope:

```
report_scopes
         VHDL Instance: {U_DEBOUNCE_0}
         VHDL Instance: {U_DEBOUNCE_1}
         VHDL Instance: {U_SINEGEN}
         VHDL Instance: {U_FSM}
          VHDL Process: {line__138}
          VHDL Process: {line__184}
          VHDL Process: {line__185}
          VHDL Process: {line__186}
          VHDL Process: {line__187}
          VHDL Process: {line__191}
```

See Also

- current_scope
- get_scopes

report_seu

Display all the SEEP changes on the device.

Syntax

`report_seu [-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

Many devices, especially those that operate in space or in high altitudes, must be able to handle single-event upsets (SEUs), caused when charged particles collide with electronic circuitry.

Xilinx IP cores, like the Soft Error Mitigation (SEM) core, and design techniques such as triple module redundancy (TMR), let you design layers of SEU protection into your products. If one circuit fails, redundant circuits perform a voting function to identify and correct the circuit that has failed. See http://www.xilinx.com/products/quality/single-event-upsets.htm for more information.

This command reports the device resources used for mitigation of single-event upsets (SEU) and other single-event effects (SEEs) in the design.

This command returns the SEU report, or returns an error if it fails.

Arguments

-file *arg* - (Optional) Write the report into the specified file. This command will overwrite any files of the same name without warning.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 SEU report to the specified file:

```
report_seu -file C:/Data/seu_mitigation.txt
```

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
path	Specify the path for pre-compiled libraries

Categories

Simulation

Description

Report information on libraries compiled by the **compile_simlib** command.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - (Required) The path to the compiled simulation library.

Examples

The following example reports information related to the compiled simulation library at the specified path:

```
report_simlib_info C:/Data/compiled_simlib
```

See Also

compile_simlib

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. The SSN predictor incorporates I/O bank-specific electrical characteristics into the prediction to better model package effects on SSN.

The **report_ssn** command can be affected by the temperature grade of the selected device as defined by the **-grade** option of the **set_operating_condition** command. Setting the temperature grade prior to running noise analysis lets you see how noisy signals can be on Commercial, Extended, Industrial, Q-Grade, or Military grade devices.

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 Virtex-7, Kintex-7, and Artix-7 devices. 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 | TXT] - (Optional) Specifies the format of the output as either comma-separated values (CSV), HTML, or an ASCII (TXT) file. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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]
[-toggle_rate] [-default_static_probability] [-default_toggle_rate]
[-file arg] [-return_string] [-append] [-quiet] [-verbose]
[objects...]
```

Returns

Nothing

Usage

Name	Description
[**-static_probability**]	report static probability
[**-signal_rate**]	report signal rate
[**-toggle_rate**]	report toggle rate
[**-default_static_probability**]	report default static probability
[**-default_toggle_rate**]	report default toggle rate
[**-file**]	Filename to output results to. (send output to console if -file is not used)
[**-return_string**]	return switching activity as string
[**-append**]	append switching activity to end of file
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	objects

Categories

Report

Description

This command is used to report different kinds of switching activity on design nets, ports, pins, and cells in the current synthesized or implemented design. 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, or the current design.

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.

-default_static_probability *value* - (Optional) Reports the default static probability to be used in power analysis on the current design. The default static probability is set using the **set_switching_activity** command.

NOTE: This option does not require objects to be specified since the default applies to the current design

-default_toggle_rate *rate* - (Optional) Reports the default toggle rate to be used in power analysis on the primary inputs of the current design. You can define the default toggle rate using the **set_switching_activity** command.

NOTE: This option does not require objects to be specified since the default applies to the current design

-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.

-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) Returns the data as a text string for assignment to a Tcl variable.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_power
- reset_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] [-no_header] [-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] [-warn_on_violation]
[-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**]	Report max delay timing paths (equivalent to -delay_type max)
[**-hold**]	Report min delay timing paths (equivalent to -delay_type min)
[**-max_paths**]	Maximum number of paths to output when sorted by slack, or per path group when sorted by group: Value >=1 Default: 1
[**-nworst**]	List up to N worst paths to endpoint: Value >=1 Default: 1

Name	Description
[**-unique_pins**]	for each unique set of pins, show at most 1 path per path group
[**-path_type**]	Format for path report: Values: end, summary, short, full, full_clock, full_clock_expanded Default: full_clock_expanded
[**-input_pins**]	Show input pins in path
[**-no_header**]	Show input pins in path
[**-label_reused**]	Label reuse status on pins in the report
[**-slack_lesser_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)
[**-sort_by**]	Sorting order of paths: Values: group, slack Default: slack
[**-no_report_unconstrained**]	Do not report infinite slack paths
[**-user_ignored**]	Only report paths which have infinite slack because of set_false_path or set_clock_groups timing constraints
[**-match_style**]	Style of pattern matching, valid values are ucf, sdc Default: sdc
[**-of_objects**]	Report timing for these paths
[**-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
[**-name**]	Output the results to GUI panel with this name
[**-return_string**]	return report as string
[**-warn_on_violation**]	issue a critical warning when the report contains a timing violation
[**-quiet**]	Ignore command errors
[**-verbose**]	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 per endpoint 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.

-no_header - (Optional) Do not write a header to the report.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

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] [-unique_pins] [-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] [-warn_on_violation] [-datasheet] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-check_timing_verbose**]	produce a verbose report when checking the design for potential timing problems
[**-delay_type**]	Type of path delay: Values: max, min, min_max Default: min_max
[**-no_detailed_paths**]	do not report timing paths for each clock and path group analyzed
[**-setup**]	Report max delay timing paths (equivalent to -delay_type max)
[**-hold**]	Report min delay timing paths (equivalent to -delay_type min)
[**-max_paths**]	Maximum number of paths to report per clock or path group: Value >=1 Default: 1
[**-nworst**]	List up to 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
[**-path_type**]	Format for path report: Values: end summary short full full_clock full_clock_expanded Default: full_clock_expanded
[**-label_reused**]	Label reuse status on pins in the report
[**-input_pins**]	Show input pins in path
[**-slack_lesser_than**]	Display paths with slack less than this Default: 1e+30
[**-report_unconstrained**]	report unconstrained paths
[**-significant_digits**]	Number of digits to display: Range: 0 to 3 Default: 3
[**-no_header**]	do not generate a report header

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
[**-warn_on_violation**]	issue a critical warning when the report contains a timing violation
[**-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.

-unique_pins - (Optional) Only report timing paths through each unique set of pins, reporting one path per path group.

-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_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.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- 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
[**-file**]	Output file
[**-append**]	Append the results to file
[**-return_string**]	return report as string
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-slr] [-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
[**-slr**]	SLR wise utilization of resources
[**-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 one 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.

-slr - (Optional) Reports the utilization for each separate SLR in devices having SLRs.

-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.

-hierarchical - (Optional) Reports the utilization of the device broken down according to the hierarchy of the design.

-hierarchical_depth *arg* - (Optional) Specifies the depth of the hierarchy to report when reporting utilization according to the hierarchy. The default depth is 0, which means that **-hierarchical** will only report the top-level by default.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 collectively utilized by all the Pblocks in the design, and writes the results to the specified file:

```
report_utilization -pblocks [get_pblocks] -file C:/Data/pblocks_util.txt
```

The following example reports the resources utilized by each Pblock in the design, appending the report for each Pblock to a single specified file:

```
foreach x [get_pblocks] {
  puts "Reporting Pblock: $x ---------------------------------"
  report_utilization -append -file C:/Data/pblocks_util.txt -pblocks $x
}
```

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
[**-radix**]	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.
[**-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

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_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.



TIP: *If no name is specified, all open DRC results windows will be closed*

This command operates silently, returning nothing if successful, or returning an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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`**`] [`*`checks...`*`]`

Returns

Nothing

Usage

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

checks - (Required) The list of one or more DRC rule checks 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_axi

Reset hardware AXI core state.

Syntax

`reset_hw_axi` `[`**`-quiet`**`]` `[`**`-verbose`**`]` `[`*`hw_axis`*`...]`

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_axis*]	List of hardware AXI objects.

Categories

Hardware

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
[**-reset_compare_values**]	Reset associated hardware probe compare values.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_ilas*]	List of hardware ILA objects. Default: Current hardware ILA

Categories

Hardware

Description

Reset the trigger and capture configuration properties on the specified ILA debug core, and the TRIGGER_COMPARE_VALUE and CAPTURE_COMPARE_VALUE properties on the core's debug probes.

Properties of the hw_ila object are configured with the **set_property** command in preparation for the **run_hw_ila** command to configure the ILA core on the hw_device. This command restores the user-configurable properties on the specified hw_ila to their default settings. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on these properties.

The default properties are:

- CONTROL.DATA_DEPTH is set to the MAX_DATA_DEPTH of the hw_ila object.
- CONTROL.TRIGGER_POSITION 0
- CONTROL.WINDOW_COUNT 1
- CONTROL.TRIGGER_MODE BASIC_ONLY
- CONTROL.TRIGGER_CONDITION AND
- CONTROL.TRIG_OUT_MODE DISABLED
- CONTROL.CAPTURE_MODE ALWAYS
- CONTROL.CAPTURE_CONDITION AND
- TRIGGER_COMPARE_VALUE eq1'bX (on the hw_probes)
- CAPTURE_COMPARE_VALUE eq1'bX (on the hw_probes)

This command operates silently, returning nothing if successful, or returning an error if it fails.

Arguments

-reset_compare_values [true | false] - (Optional) Reset the TRIGGER_COMPARE_VALUE and CAPTURE_COMPARE_VALUE properties on the hw_probes associated with the specified hw_ila object. This is a boolean argument that is TRUE, or enabled, by default. If **-reset_compare_values false** is used, the compare value properties on the probes are not reset. In this case, the properties on the hw_ila are reset, but the properties on the hw_probes are left as currently configured.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_ilas - (Optional) Specify one or more hw_ila objects to reset. The hw_ila objects can either be specified as objects returned by the **get_hw_ilas** or **current_hw_ila** commands, or specified by name. If the hw_ila is not specified, the **current_hw_ila** will be reset.

Example

The following example resets all hw_ila debug cores on the current device:

```
reset_hw_ila [get_hw_ilas]
```

See Also

- current_hw_ila
- get_hw_ilas
- run_hw_ila
- set_property

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

Description

Resets the ACTIVITY_VALUE properties for all hardware probes on the specified VIO debug core objects. The ACTIVITY_VALUE property is used by the Vivado IDE to represent transitions on the input probes of the VIO debug cores.

In addition to reading values from the VIO input probes, you can also monitor the activity of the VIO input probes. The ACTIVITY_VALUE property is used to indicate when the values on the VIO inputs have changed in between periodic updates to the Vivado IDE. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_vios - (Required) Specify one or more hw_vio objects to reset. The hw_vio objects can either be specified as objects returned by the **get_hw_vios** command, or specified by name.

Example

The following example resets the input activity properties of the VIO debug core:

```
reset_hw_vio_activity [get_hw_vios]
```

See Also

- commit_hw_vio
- connect_hw_server
- current_hw_device
- get_hw_probes
- get_hw_vios
- program_hw_devices
- refresh_hw_vio
- reset_hw_vio_outputs
- set_property

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

Description

Reset the hardware VIO debug core outputs to their initial, or "reset" state.

The Virtual Input/Output (VIO) debug core can both monitor and drive internal signals on a programmed Xilinx FPGA device in real time. The VIO core uses hardware probes, hw_probe objects, to monitor and drive signals on the device. Input probes monitor signals as inputs to the VIO core. Output probes drive signals to specified values from the VIO core.

The reset_hw_vio_outputs command restores the signal values at the output probes of the specified hw_vio debug cores to their initial values. This affects the signal on the hw_device, but does not affect the OUTPUT_VALUE property of the hw_probe objects.



TIP: *This command has the effect of resetting the initial value of the signal on the hw_vio debug core, without resetting the properties on the hw_probe object*

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_vios - (Required) Specify one or more hw_vio objects to reset. The hw_vio objects can either be specified as objects returned by the **get_hw_vios** command, or specified by name.

Example

The following example resets the output probes on the VIO debug core to the initial values on the core when the FPGA device was first configured and booted:

```
reset_hw_vio_outputs [get_hw_vios {hw_vio_1}]
```

See Also

- commit_hw_vio
- connect_hw_server
- current_hw_device
- get_hw_probes
- get_hw_vios
- program_hw_devices
- refresh_hw_vio
- reset_hw_vio_activity
- set_property

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] [-regexp] [-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
[**-regexp**]	The values used for -string are full regular expressions
[**-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 severity of the specified message ID to a Critical Warning, and then resets the message to its default severity:

```
set_msg_config -id "Common 17-81" -new_severity "CRITICAL WARNING"
reset_msg_config -id "Common 17-81" -default_severity
```

The following example suppresses the specified message ID using **set_msg_config**, and then un-suppresses it using **reset_msg_config**:

```
set_msg_config -id {HDL 9-1654} -suppress
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
id	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

set_msg_config

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 returns nothing if successful, or returns an error if it fails.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-exclude_runs**]	Do not reset runs
[**-exclude_ips**]	Do not reset ips
[**-exclude_sim_runs**]	Do not reset simulation runs
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
property_name	Name of property to reset
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 sets the DOB_REG property on the specified Block RAM, and then resets the property:

```
set_property DOB_REG 1 [get_cells usbEngine1/usbEngineSRAM/snoopyRam_reg_19]
reset_property DOB_REG [get_cells usbEngine1/usbEngineSRAM/snoopyRam_reg_19]
```

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
[**-prev_step**]	Reset last run step
[**-from_step**]	First Step to reset
[**-noclean_dir**]	Do not remove all output files and directories from disk
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
run	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Description

Reset the current simulation to its starting condition, by cleaning out the various output files created during compilation and simulation for the specified simulation fileset.



IMPORTANT: *Local files will be removed from the project simulation folders without warning*

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

Arguments

-mode [behavioral | post-synthesis | post-implementation] - (Optional) Specify the simulation mode to reset. Valid values include behavioral simulation, post-synthesis, or post implementation simulation. 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. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

simset - (Optional) Specify a simulation fileset to remove. The default simset is sim_1.

Examples

The following example resets the post-synthesis timing simulation by removing files for the sim_2 simset:

```
reset_simulation -mode post-synthesis -type timing sim_2
```

See Also

launch_xsim

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
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 [-default] [-hier] [-quiet] [-verbose]
[objects...]
```

Returns

Nothing

Usage

Name	Description
[**-default**]	Reset default static probability and default toggle rate
[**-hier**]	Hierarchically resets the switching activity on a hierarchical cells provided as *objects*.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	Objects to reset switching activity on

Categories

Power

Description

Resets the attributes of the switching activity on specified nets, ports, pins, and cells in the design.

The switching activity is 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.

NOTE: The **reset_switching_activity** is used to reset switching activity for specified objects. Use **set_switching_activity -default_toggle_rate** or **-default_static_probability** to change or reset the default values for the current design

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

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 list of objects for which to reset the switching activity. If not specified, the command resets the switching activity on all objects.

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_power
- report_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
name	List of targets to be reset, or 'all' to reset all generated targets
objects	The objects for which data needs to be reset

Categories

Project, 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 rest 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

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
[**-from**]	New starting bus index
[**-to**]	New ending bus index
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
net_bus_name	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 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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 renumbers 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, add and remove Pblock site-range constraints.

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
pblock	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-from**]	New starting bus index
[**-to**]	New ending bus index
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
pin_bus_name	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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-from**]	New starting bus index
[**-to**]	New ending bus index
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
port_bus_name	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 the elaborated 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 renumbers 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Simulation

Description

Return the current simulation to its initial state, as if the design was reloaded, resetting the current simulation time to 0.

The **restart** command retains breakpoints, Tcl forces, and settings in the waveform configuration window, but resets all simulation values, and clears all other Tcl commands.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 restarts the current simulation:

```
restart
```

See Also

- current_time
- run
- stop

route_design

Route the current design.

Syntax

```
route_design [-unroute] [-release_memory] [-nets args] [-physical_nets]
[-pin arg] [-directive arg] [-tns_cleanup] [-no_timing_driven]
[-preserve] [-delay] -max_delay arg -min_delay arg [-timing_summary]
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-unroute**]	Unroute whole design or the given nets/pins if used with -nets or -pin.
[**-release_memory**]	Release Router memory. Not compatible with any other options.
[**-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
[**-tns_cleanup**]	Do optional TNS clean up.
[**-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.
-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.
[**-timing_summary**]	Enable post-router signoff timing summary.
[**-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 options of the **route_design** command to achieve route completion and timing closure. 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.

-release_memory - (Optional) Free router memory resources for subsequent route passes. This option does not run route passes, but only releases memory held by the router to reduce router initialization. The router will need to reload design data for subsequent route passes.

-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**, **-physical_nets**, or **-pin** specified. Use **-delay** to find a route with the shortest delay.

-physical_nets - (Optional) Route or unroute only logic zero and logic one nets.

-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.

-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, except **-tns_cleanup**. It can also 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**

-tns_cleanup - (Optional) By default, to reduce runtime, the router focuses on optimizing the Worst Negative Slack (WNS) path as opposed to Total Negative Slack (TNS) paths. This option invokes an optional phase at the end of routing where the router attempts to fix the TNS paths, those failing paths other than the WNS path. This option may reduce TNS at the cost of added runtime, but will not affect WNS. The **-tns_cleanup** option is recommended when using post-route **phys_opt_design** to ensure that optimization focuses on the WNS path and does not waste effort on TNS paths that can be fixed by the router. This option can be used in combination with **-directive**.

-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.

-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.

-timing_summary - (Optional) Generate the sign-off timing summary report after routing has completed. The sign-off timing summary report is the same as the **report_timing_summary** command. By default the router performs an estimated timing analysis based on incremental router updates during the design implementation.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
```

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
[*time*]	Length of simulation time
[*unit*]	Unit for time from the following time units: fs, ps, ns, us, ms, sec

Categories

Simulation

Description

Run the current simulation from the current time to the specified time, or until the simulation stops.

A running simulation can be stopped at a predetermined time, at a specific breakpoint in the HDL source code, by encountering a TRUE condition, by evaluating the circuit until there are no remaining events, or by encountering a runtime error such as an out-of-bounds value.

The **run** command instructs an existing simulation to run for a specified length of time, or until there are no remaining events. The time is specified as a floating point number indicating a period of time in the current simulation units, or in the specified units.

Arguments

-all - Run the simulation until no event is left in the event queue, a breakpoint or valid condition is encountered, or a run time exception occurs.

NOTE: This is the default when no other options are specified

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

time - (Optional) Specifies the length of the simulation run time. The time can be specified as a floating point number, or using the keyword **all**.

NOTE: Using the keyword **all** is the same as using the **-all** option

unit - (Optional) One of the following time units (with or without space between time and unit) can be specified: **fs**, **ps**, **ns**, **us**, **ms**, or **sec**. The default is the value of **time_unit** .

Examples

The following example runs an existing simulation for the specified simulation run time, using the default units (ns):

```
run 1000
```

The following example runs an existing simulation for 300 microseconds (us):

```
run 300 us
```

The following example runs the current simulation until no event is left in the event queue, a breakpoint or valid condition is met, or a simulation runtime error occurs:

```
run -all
```

See Also

- add_bp
- add_condition
- restart
- step
- stop

run_hw_axi

Run hardware AXI read/write transaction(s)and update transaction status in hw_axi object..

Syntax

run_hw_axi [**-quiet**] [**-verbose**] *hw_axi_txns* ...

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_axi_txns	hardware AXI Transaction object to execute on the AXI bus.

Categories

Hardware

run_hw_ila

Arm hardware ILAs.

Syntax

run_hw_ila [**-trigger_now**] [**-compile_only**] [**-file** *arg*] [**-force**] [**-quiet**] [**-verbose**] [*hw_ilas...*]

Returns

Nothing

Usage

Name	Description
[**-trigger_now**]	Trigger and capture immediately.
[**-compile_only**]	Test only compile trigger state machine file but do not upload.
[**-file**]	Trigger at startup file name. Command will not arm ILA core.
[**-force**]	Overwrite existing file
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_ilas*]	hardware ILAs Default: Current hardware ILA

Categories

Hardware

Description

Arm triggers and run the specified hardware ILA debug cores on the current hardware device.

The Integrated Logic Analyzer (ILA) debug core lets you perform in-system debug of implemented designs, or design bitstreams, on a programmed Xilinx FPGA device. The ILA core includes many advanced features of modern logic analyzers, including boolean trigger equations, and edge transition triggers. You can use the ILA core to probe specific signals of the design, to trigger on programmed hardware events, and capture data samples from the Xilinx FPGA device in real-time. Refer to *LogiCORE IP Integrated Logic Analyzer (PG172)* for details of the ILA core.

You can add ILA debug cores into the RTL source files of a design, or in the synthesized netlist using the **create_debug_core** command. Refer to the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)* for more information on adding debug cores and signal probes to the design.

Debug cores and probes are written to a probes file (.ltx) using the **write_probes_file** command, and associated with the hardware device along with the bitstream file (.bit) using the PROBES.FILE and PROGRAM.FILE properties of the hw_device object. The hardware device is programmed using the **program_hw_devices** command. The ILA debug cores in the design are accessible from the Hardware Manager using the **get_hw_ilas** command. The debug probes assigned to the ILA debug cores can be returned with the **get_hw_probes** command.

The steps to debug your design in hardware using an ILA debug core are:

1. Connect to the hardware server and target using **connect_hw_server** and **open_hw_target**.
2. Program the FPGA device with the bitstream (.bit) and probes (.ltx) files using **program_hw_devices**.
3. Set up the ILA debug core trigger events and data capture controls using **set_property** to configure properties of the ILA.
4. Arm the ILA debug core triggers on the Xilinx FPGA using **run_hw_ila**. When a trigger event occurs, or the capture condition is met, the ILA capture buffer is filled.
5. Uploaded sampled data from the hw_device into a Vivado logic analyzer hw_ila_data object using **upload_hw_ila_data**.
6. View the captured data in the Waveform window of the Vivado logic analyzer feature using **display_hw_ila_data**.

You can set up an ILA debug core to trigger on specific events or conditions at the debug probes, and to capture data under specific conditions, using CONTROL properties on the hw_ila object. You set these properties with the **set_property** command. Refer to the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)* for more information on setting properties to configure debug cores and signal probes to monitor the design.

RECOMMENDED: *The Vivado IDE provides a graphical interface to configure hw_ila and hw_probes for trigger and capture. You can use the Vivado IDE to see the properties needed to configure and run the hw_ila*

The specific properties on the hw_ila you can use to configure a debug core include the following:

- CONTROL.DATA_DEPTH - Defaults to the MAX_DATA_DEPTH of the ILA debug core, which was set when the debug core was created or inserted into the design. The data depth defines the number of data samples the hw_ila object can capture in a data window. Set the data depth as an integer which is a power of two, from 1 to the maximum data depth (MAX_DATA_DEPTH) of the hw_ila.

  NOTE: The value of DATA_DEPTH is related to CONTROL.WINDOW_COUNT by the equation: **DATA_DEPTH * WINDOW_COUNT = MAX_DATA_DEPTH**

- CONTROL.WINDOW_COUNT - Lets you divide the MAX_DATA_DEPTH of the ILA core into a number of data windows to store sample data from multiple trigger events. In the case of 10 data windows for example, the first window will be filled at the first trigger event, and each subsequent window will be filled upon subsequent triggering events or capture conditions.

- CONTROL.TRIGGER_POSITION - An integer value related to the DATA_DEPTH. Positions the trigger event in the sample data buffer. For a DATA_DEPTH of 1024, position 0 refers to the first (or left-most) data buffer and 1023 refers to the last (or right-most) data buffer. The TRIGGER_POSITION lets you capture sample data ahead of the trigger event. For instance, with a DATA_DEPTH of 256, a TRIGGER_POSITION of 100 allows you to capture 100 data samples ahead of the trigger, and 155 data samples at and after the trigger event.

  NOTE: In the case of **run_hw_ila -trigger_now 1**, the TRIGGER_POSITION merely positions the trigger mark in the Vivado logic analyzer waveform window. Because the trigger event is immediate, there is no time to capture data samples ahead of the trigger

- CONTROL.TRIGGER_MODE - Valid values include:
  - BASIC_ONLY - The trigger condition is the result of a Boolean equation using the TRIGGER_CONDITION to evaluate the values on each of the associated ILA probes.
  - BASIC_OR_TRIG_IN - The ILA core is triggered by a Boolean equation considering probe values, or by the TRIG_IN port on the core.
  - ADVANCED_ONLY - The ILA core is configured to have advanced trigger capabilities defined in a user-defined Trigger State Machine (TSM).
  - ADVANCED_OR_TRIG_IN - The ILA core is triggered by the TSM or by the TRIG_IN port on the core.
  - TRIG_IN_ONLY - The ILA core is triggered only by the TRIG_IN port on the core.

- CONTROL.TRIGGER_CONDITION - Defines a Boolean equation which evaluates comparators on participating probes on the ILA debug core. When the condition evaluates to true, the BASIC trigger mode is satisfied. Valid values include:
  - AND - Trigger condition is "true" if all participating probe comparators evaluate "true", otherwise trigger condition is "false."
  - NAND - Trigger condition is "true" if at least one participating probe comparator evaluates "false", otherwise trigger condition is "false."
  - OR - Trigger condition is "true" if at least one participating probe comparator evaluates "true", otherwise trigger condition is "false."
  - NOR - Trigger condition is "true" if all participating probe comparators evaluate "false", otherwise trigger condition is "false."

  NOTE: The evaluation of the probes participating in the trigger condition is determined by the TRIGGER_COMPARE_VALUE property assigned to the hw_probe object, as returned by **get_hw_probes**. If the TRIGGER_COMPARE_VALUE is 'X' then it is not participating in the trigger condition

- CONTROL.TSM_FILE - Specify the path to a file defining a Trigger Finite State Machine (TSM) to be used for advanced trigger handling.
- CONTROL.TRIG_OUT_MODE - Used to transition the TRIG_OUT port on the ILA core to be used to drive the TRIG_IN port on other ILA cores. Valid values include:
  - DISABLED - Disable the TRIG_OUT port on the ILA core.
  - TRIGGER_ONLY - Transition the TRIG_OUT port when the trigger conditions have been satisfied.
  - TRIG_IN_ONLY - Transition the TRIG_OUT when the TRIG_IN signal transitions. Use this to pass the trigger event along a chain of ILA cores.

  - TRIGGER_OR_TRIG_IN - Transition the TRIG_OUT when either the trigger conditions are satisfied, or the TRIG_IN transitions.

- CONTROL.CAPTURE_MODE - Valid values include ALWAYS or BASIC. Capture and store a data sample on the debug core ALWAYS during a given clock cycle, or only if the CAPTURE_CONDITION evaluates to "true" (BASIC).
- CONTROL.CAPTURE_CONDITION - Defines a Boolean equation for participating probe comparators on the ILA debug core that must evaluate to TRUE to meet the data capture condition. When the capture condition evaluates to true, the BASIC capture mode is satisfied. Valid values include:
  - AND - Capture condition is "true" if all participating probe comparators evaluate "true", otherwise capture condition is "false."
  - NAND - Capture condition is "true" if at least one participating probe comparator evaluates "false", otherwise capture condition is "false."
  - OR - Capture condition is "true" if at least one participating probe comparator evaluates "true", otherwise capture condition is "false."
  - NOR - Capture condition is "true" if all participating probe comparators evaluate "false", otherwise capture condition is "false."

  NOTE: The evaluation of the probes participating in the capture condition is determined by the CAPTURE_COMPARE_VALUE property assigned to the hw_probe object, as returned by **get_hw_probes**. If the CAPTURE_COMPARE_VALUE is 'X' then it is not participating in the trigger condition.



TIP: *There are other properties on the ILA core that also determine the operation of the core, but they are not user-configurable*

With the ILA core configured, you can use the **run_hw_ila** command to arm the ILA cores on the target part. When this command is run, the trigger configurations defined in the hw_ila and hw_probe objects are written to the target Xilinx FPGA (hw_device) and arms the ILA core or cores on the device.

With the hw_ila armed and running, the **wait_on_hw_ila** command stops your Tcl script to wait for the data sample buffers to be populated with captured data. When the memory of the ILA core is full on the physical hw_device, the **wait_on_hw_ila** command returns, and your Tcl script resumes.

You can use **upload_hw_ila_data** to upload the captured data from the physical memory of the hw_device into a hw_ila_data object in the Vivado logic analyzer. Then view the ILA data in the waveform window of the Vivado logic analyzer using **display_hw_ila_data**, and write the data for use in external tools using the **write_hw_ila_data** command.

You can also immediately trigger the probes on the hw_device using the **-trigger_now** option, to capture data from the device right away, rather than waiting for trigger events or capture conditions to be met over time.

You can use **reset_hw_ila** to restore the CONTROL properties of the ILA debug core to their default setting, and reset the probe comparator values to 'X'.

Arguments

-trigger_now - (Optional) Immediately trigger the ILA debug cores, regardless of trigger conditions, to capture sample data at the debug ports. This is a boolean argument enabled by its presence.

-compile_only - (Optional) Compile the trigger state machine file (CONTROL.TSM_FILE) to perform syntax checking and report compiler errors and warnings. This option does not load and arm the hardware device. This option is only supported when in advanced trigger mode (CONTROL.TRIGGER_MODE ADVANCED_ONLY), and will otherwise return an error.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 *arg* - (Optional) When the TRIGGER_MODE is set to BASIC_ONLY, this option will export a file containing the current trigger and probe configuration settings. The trigger file can be used by the **apply_hw_ila_trigger** command to define triggers for the hw_device to use on startup, or upon booting the device. Using the **-file** option prevents the **run_hw_ila** command from arming the ILA core triggers on the hardware device.

NOTE: If no extension is specified, a default file extension of `.tas` is assigned

-force - (Optional) Overwrite the specified trigger file if it already exists. By default the tool will not overwrite an existing file.

hw_ilas - (Optional) Specify one or more hw_ila objects to run. The hw_ila objects can either be specified as objects returned by the **get_hw_ilas** or **current_hw_ila** commands, or specified by name. If the hw_ila is not specified, the **current_hw_ila** will be run.



IMPORTANT: *You can only specify one hw_ila object when using the **-file** option*

Example

The following example sets the trigger and capture properties on the current hw_ila object, and then runs the hw_ila, specified by name, to arm the hw_device and wait for the data buffers to be filled:

```
current_hw_ila [get_hw_ilas hw_ila_1]
set_property CONTROL.DATA_DEPTH 1024 [current_hw_ila]
set_property CONTROL.TRIGGER_MODE BASIC_ONLY [current_hw_ila]
set_property CONTROL.TRIGGER_CONDITION AND [current_hw_ila]
set_property CONTROL.CAPTURE_MODE ALWAYS [current_hw_ila]
run_hw_ila -trigger_now hw_ila_1
wait_on_hw_ila hw_ila_1
```

This example writes a trigger configuration file for the current hw_ila object, to the specified file, but does not arm the current hw_device:

```
run_hw_ila -file C:/Data/trigger_config1
```

NOTE: A default file extension of `.tas` is assigned to the specified file

See Also

- apply_hw_ila_trigger
- current_hw_device
- current_hw_ila
- current_hw_ila_data
- get_hw_devices
- get_hw_ilas
- get_hw_ila_datas
- get_hw_probes
- reset_hw_ila
- set_property
- upload_hw_ila_data
- wait_on_hw_ila
- write_debug_probes
- write_hw_ila_data

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
hw_sio_scans	hardware SIO scans

Categories

Hardware

Description

Run the specified serial I/O analyzer link scan.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized Eye Scan hardware of Xilinx UltraScale devices or 7 Series FPGAs. The Vivado serial I/O analyzer feature lets you to create, run, and save link scans.

This command creates and returns a link scan object that you can use with the **run_hw_sio_scan** command to run analysis on the specified links, or GT receivers. You can also save the scan to disk using the **write_hw_sio_scan** command.

This command run analysis on the specified scan objects. If running in a Tcl script, you can suspend the script while the scan completes using the **wait_on_hw_sio_scan** command. You can stop a running scan using the **stop_hw_sio_scan** command.

You can save the scan to disk using the **write_hw_sio_scan** command.

You can remove the created scan object using **remove_hw_sio_scan**.

This command returns the hw_sio_scan object, or returns an error if the command fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_scans - (Required) Specify one or more hw_sio_scan objects to run. The hw_sio_scans must be specified as objects as returned by the **create_hw_sio_scan** or **get_hw_sio_scans** commands.

Example

The following example runs the specified serial I/O analyzer scan:

```
run_hw_sio_scan [lindex [get_hw_sio_scans {SCAN_3}] 0]
```

See Also

- create_hw_sio_scan
- current_hw_device
- get_hw_sio_scans
- remove_hw_sio_scan
- run_hw_sio_scan
- stop_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

run_hw_sio_sweep

Run hardware SIO sweeps.

Syntax

`run_hw_sio_sweep` `[`**`-quiet`**`]` `[`**`-verbose`**`]` *`hw_sio_sweeps`*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_sio_sweeps	hardware SIO sweeps

Categories

Hardware

Description

Run a serial I/O analyzer link sweep scan to run multiple scans across a range of values.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized features of Xilinx UltraScale devices or 7 Series FPGAs. It can also be helpful to run multiple scans on a the link with different configuration settings for the GTs. This can help you determine which settings are best for your design. The Vivado serial I/O analyzer feature enables you to define, run, and save link sweeps, or collections of link scans run across a range of values.

This command run analysis on the specified sweep scan objects. If running in a Tcl script, you can suspend the script while the sweep scan completes using the **wait_on_hw_sio_sweep** command. You can stop a running sweep scan using the **stop_hw_sio_sweep** command.

You can save the sweep scan to disk using the **write_hw_sio_sweep** command.

You can remove the created scan object using **remove_hw_sio_sweep**.

This command returns the hw_sio_sweep object, or returns an error if the command fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_sweeps - (Required) Specify one or more hw_sio_sweep objects to run. The hw_sio_sweep must be specified as an object as returned by the **get_hw_sio_sweeps** command.

Example

The following example runs the specified sweep scan:

```
run_hw_sio_sweep [lindex [get_hw_sio_sweeps {SWEEP_0}] 0]
```

See Also

- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_sweeps
- remove_hw_sio_sweep
- stop_hw_sio_sweep
- wait_on_hw_sio_sweep
- write_hw_sio_sweep

run_state_hw_jtag

Change to a stable state of a specified transition.

Syntax

`run_state_hw_jtag` [**`-state`** *`args`*] [**`-quiet`**] [**`-verbose`**] *`stable_state`*

Returns

Hardware JTAG

Usage

Name	Description
[**-state**]	valid state path sequence to stable_state
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
stable_state	valid stable_state - valid stable states IDLE, RESET, IRPAUSE, and DRPAUSE

Categories

Hardware, Object

runtest_hw_jtag

Forces IEEE 1149.1 TAP state machine to a stable state for a specified wait period.

Syntax

```
runtest_hw_jtag [-wait_state arg] [-end_state arg] [-sec arg]
[-max_wait arg] [-tck arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-wait_state**]	valid stable_state - valid stable states IDLE, RESET, IRPAUSE, and DRPAUSE
[**-end_state**]	valid stable_state - valid stable states IDLE, RESET, IRPAUSE, and DRPAUSE
[**-sec**]	Number of seconds to wait in wait_state
[**-max_wait**]	Maximum Number of seconds to wait in wait_state - max timeout
[**-tck**]	Number of TCK cycles to wait in wait_state Default: Number of TCK cycles to wait in wait_state
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Hardware, Object

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
[*name*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-dir**]	Directory to save constraints to
[**-target_constrs_file**]	Target constraints file for the new fileset
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	New name for the project to save
[*dir*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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*.xpr 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*I 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
[*filename*]	Filename to save the specified or current wave configuration object

Categories

Waveform

Description

Save the current or specified wave configuration object to a specified filename.

If the wave configuration object has not been saved before, and does not have a FILE_PATH property value, the *filename* is required and the NAME of the wave configuration object will be changed to match the specified *filename*.

If the specified wave configuration object has been previously saved, and has a FILE_PATH property, the object will be written to its current location, and the *filename* does not need to be specified.

If the wave configuration object has a FILE_PATH property, but a different *filename* is specified, the wave configuration object will be saved to the new *filename*, and the object will be renamed to match the specified *filename*.

Arguments

-object *arg* - (Optional) Specify a wave configuration object to save. If the wave configuration object has not been saved to a file before, you must also specify the *filename*. If **-object** is not specified, the current wave configuration object , as returned by **current_wave_config** is saved to the specified *filename*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

filename - (Optional) Specify a path and *filename* to save the current or specified wave configuration object. The *filename* should have a suffix of `.wcfg`, and the file suffix will be assigned automatically if no other suffix is supplied.

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 saves the specified wave configuration object to a new filename:

```
save_wave_config -object [get_wave_configs test.wcfg] \
C:/Data/project/newTest
```

NOTE: The wave config file will be assigned the `.wcfg` suffix since none is specified

See Also

- create_wave_config
- current_wave_config
- get_wave_configs
- open_wave_config

scan_dr_hw_jtag

Perform shift DR on 'hw_jtag'.

Syntax

scan_dr_hw_jtag [**-tdi** *arg*] [**-tdo** *arg*] [**-mask** *arg*] [**-smask** *arg*] [**-quiet**] [**-verbose**] *length*

Returns

Hardware TDO

Usage

Name	Description
[**-tdi**]	Hex value to be scanned into the target
[**-tdo**]	Hex value to be compared against the scanned value
[**-mask**]	Hex value mask applied when comparing TDO values
[**-smask**]	Hex value mask applied to TDI value
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
length	Number of bits to be scanned.

Categories

Hardware, Object

scan_ir_hw_jtag

Perform shift IR on 'hw_jtag'.

Syntax

scan_ir_hw_jtag [**-tdi** *arg*] [**-tdo** *arg*] [**-mask** *arg*] [**-smask** *arg*] [**-quiet**] [**-verbose**] *length*

Returns

Hardware TDO

Usage

Name	Description
[**-tdi**]	Hex value to be scanned into the target
[**-tdo**]	Hex value to be compared against the scanned value
[**-mask**]	Hex value mask applied when comparing TDO values
[**-smask**]	Hex value mask applied to TDI value
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
length	Number of bits to be scanned.

Categories

Hardware, Object

select_objects

Select objects in GUI.

Syntax

`select_objects` `[-add] [-quiet] [-verbose]` *`objects`*

Returns

Nothing

Usage

Name	Description
[**-add**]	Add to existing selection list
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
- highlight_objects
- mark_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
value	Logic value on the pin: Values: 0, 1, rising, falling, zero, one, rise, fall
objects	List of ports or pins

Categories

SDC, XDC

Description

Specifies that a pin or port is in a steady state of 1, 0, rising or falling.

This command is usually used to force values onto the ports to help reduce the analysis space, runtime and memory consumption. It is important to let the Vivado timing engine know about signals that have a constant value. This is also critical to ensure that non-functional and irrelevant paths are not reported.

Setting a case value on a pin results in disabling timing analysis through that pin. This means that timing paths through that pin are not reported.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. When the values ris(e)(ing) or fall(ing) are specified, the given pins or ports are only considered for timing analysis with the specified transition. The other transition is disabled. The default setting is 1.

objects - (Required) One or more ports or pins on which to apply the *value*.

Examples

In the example below, two clocks are created on the input pins of the BUFGMUX, clock_sel, but only clk_B is propagated through the output pin after setting the constant value 1 on the selection pin S:

```
create_clock -name clk_A -period 10.0 [get_pins clock_sel/I0]
create_clock -name clk_B -period 15.0 [get_pins clock_sel/I1]
set_case_analysis 1 [get_pins clock_sel/S]
```

See Also

- create_clock
- 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.

If only one group is specified, the clocks in that group are asynchronous or exclusive to all other clocks in the design, but not to each other. If a new clock is created after the set_clock_groups command, it is asynchronous to that group as well.

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.

NOTE: **-logically_exclusive**, **-physically_exclusive** and **-asynchronous** are mutually exclusive arguments.

-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.

-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.



TIP: *If only one group of clocks is specified, that group is exclusive with or asynchronous to all other clocks in the design. Clocks can be specified by name, or as clock objects returned by the* ***get_clocks*** *command*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 into separate clock groups. The clock groups are asynchronous to each other:

```
set_clock_groups -group src_clk -group sync_clk -asynchronous
```

The following example includes the generated clocks of the specified clocks, and adds those to the clock group:

```
set_clock_groups -group [get_clocks -include_generated_clocks src_clk] \
-group [get_clocks -include_generated_clocks sync_clk]  -asynchronous
```

NOTE: In the preceding example, src_clk and sync_clk, and all their generated clocks, are asynchronous. Otherwise the generated clocks would be timed against each other and the other master clock.

In this example, the specified clocks are grouped together, and are asynchronous to all other clocks in the design:

```
set_clock_groups -async -group [get_clocks {J_CLK U_CLK}]
```

See Also

- get_clocks
- 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
latency	Latency value
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
pins	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
uncertainty	Uncertainty of clock network
[*objects*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, 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
value	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 returns nothing if successful, or returns an error if it fails.

Arguments

-from *value* - (Optional) Check the datapath from the specified data pin or port. The **-from** argument specifies the related pin.

-to *value* - (Optional) Check the datapath to the specified data pin or port. The **-to** argument specifies the constrained pin

-rise_from *value* - (Optional) Check the datapath from the rising edge of the specified data pin or port.

-fall_from *value* - (Optional) Check the datapath from the falling edge of the specified data pin or port.

-rise_to *value* - (Optional) Check the datapath to the rising edge of the specified data pin or port.

-fall_to *value* - (Optional) Check the datapath to the falling edge of the specified data pin or port.

-setup *value* - (Optional) Perform only the setup data check. The default is to perform both setup and hold checks.

-hold *value* - (Optional) Perform the hold data check. The default is to perform both setup and hold checks.

-clock *value* - (Optional) Specify the clock domain at the related pin or port of the checks.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 specified in nanoseconds 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_min_delay
- set_max_delay

set_delay_model

Sets the interconnect delay model for timing analysis.

Syntax

`set_delay_model` **`[-interconnect`** *`arg`*`]` **`[-quiet]`** **`[-verbose]`**

Returns

Nothing

Usage

Name	Description
[**-interconnect**]	Interconnect delay model used for timing analysis: Values: estimated, actual(default), none
[**-quiet**]	Ignore command errors
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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** *args*] [**-to** *args*] [**-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
delay_value	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_hw_sysmon_reg

Set the system monitor register value.

Syntax

set_hw_sysmon_reg [**-quiet**] [**-verbose**] *hw_sysmon hexaddress hexdata*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_sysmon	hw_sysmon object
hexaddress	Hex address to write to
hexdata	Hex write value

Categories

Hardware

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
[**-clock**]	Relative clock
[**-reference_pin**]	Relative pin or port
[**-clock_fall**]	Delay is relative to falling edge of clock
[**-rise**]	Specifies rising delay
[**-fall**]	Specifies falling delay
[**-max**]	Specifies maximum delay
[**-min**]	Specifies minimum delay
[**-add_delay**]	Don't remove existing input delay
[**-network_latency_included**]	Specifies network latency of clock already included
[**-source_latency_included**]	Specifies source latency of clock already included
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
delay	Delay value
objects	List of ports

Categories

SDC, XDC

Description

Specifies the external system-level path delay on a primary input port relative to a clock edge at the interface of the design. The input delay value can be positive or negative, depending on the clock and data relative phase at the interface of the device.

To accurately model the system-level timing of your Xilinx FPGA design, you must assign timing delays for objects external to the FPGA onto the primary input or output ports in your design. These delays are defined by the **set_input_delay** and **set_output_delay** commands.



IMPORTANT: *If the input port also has a **set_max_delay** constraint assigned, the specified input delay value is considered part of the max_delay computation. That is, the input delay consumes a portion of the max delay on the timing path that includes the input port*

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

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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. Valid values are floating point numbers, with a default value of 0.

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_max_delay
- 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
clock	Clock
input_jitter	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 returns nothing if successful, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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/sysClkDiv
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
capacitance	Capacitance value
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-rise**]	Delay value applies to rising path delays
[**-fall**]	Delay value applies to falling path delays
[**-reset_path**]	Reset this path before setting max delay
[**-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
[**-datapath_only**]	Remove clock skew and jitter from calculation
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
delay	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.

The maximum rising and falling delay cannot be less than the minimum rising and falling delay on the same path. If this happens, the earlier assigned delay value is removed from the timing path as a conflict, and the delay is set 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.



IMPORTANT: *When assigned to a primary input or output port, any system-level delay consumes a portion of the max delay on the timing path that includes the input or output port. That is, the delay specified by* ***set_input_delay*** *or* ***set_output_delay*** *is considered part of the maximum delay*

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

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 startpoints.

-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.

NOTE: The hold check for the specified path is automatically defined as a false path when using the **-datapath_only** option

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 specified in nanoseconds. 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_input_delay
- set_min_delay
- set_output_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
delay	Delay value: Value >= 0
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-rise**]	Delay value applies to rising path delays
[**-fall**]	Delay value applies to falling path delays
[**-reset_path**]	Reset this path before setting min delay
[**-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
delay	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.



IMPORTANT: *The minimum rising and falling delay cannot be greater than the maximum rising and falling delay on the same path. If this happens, the first 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.

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

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 nanoseconds. The *delay* can be specified as a floating point number >= 0, with a default value of 0.

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_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] [-regexp] [-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. Can only be used in conjunction with one of -id or -severity.
[**-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
[**-regexp**]	The values used for -string are full regular expressions
[**-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_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
path_multiplier	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-clock**]	Relative clock
[**-reference_pin**]	Relative pin or port
[**-clock_fall**]	Delay is relative to falling edge of clock
[**-rise**]	Specifies rising delay
[**-fall**]	Specifies falling delay
[**-max**]	Specifies maximum delay
[**-min**]	Specifies minimum delay
[**-add_delay**]	Don't remove existing input delay
[**-network_latency_included**]	Specifies network latency of clock already included
[**-source_latency_included**]	Specifies source latency of clock already included
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
delay	Delay value
objects	List of ports

Categories

SDC, XDC

Description

Specifies the external system-level path delay on a primary output port relative to a clock edge at the interface of the design. The output delay value can be positive or negative, depending on the clock and data relative phase outside the FPGA device.

To accurately model the system-level timing of your Xilinx FPGA design, you must assign timing delays for objects external to the FPGA onto the primary input or output ports in your design. These delays are defined by the **set_input_delay** and **set_output_delay** commands.



IMPORTANT: *If the output port also has a **set_max_delay** constraint assigned, the specified input delay value is considered part of the max_delay computation. That is, the output delay consumes a portion of the max delay on the timing path that includes the output port*

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

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, with a default value of 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
column	User column name
value	Value to set
package_pins	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
name	Parameter name
value	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-include_cells**]	Include only these instances for clock gating. Default: all
[**-exclude_cells**]	Exclude these instances from clock gating. Default: none
[**-clocks**]	Clock gate instances clocked by these clocks only. Default: all clocks
[**-cell_types**]	Clock gate these types only. One of [all|bram|reg|srl|none]. Default: all. Default: all
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Power, XDC

Description

Specify cell instances to include or exclude in power optimization. The specified cells are optimized using the **power_opt_design** command.



TIP: *Block RAM optimizations are performed by default with the **opt_design** command. Some or all BRAM cells can be excluded from the **opt_design** optimization using the **set_power_opt** command as well*

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 | slr | 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

- opt_design
- 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
objects	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

Nothing

Usage

Name	Description
[**-dict**]	list of name/value pairs of properties to set
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Name of property to set. Not valid with -dict option
value	Value of property to set. Not valid with -dict option
objects	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.

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



TIP: *You can use the **get_property** command to validate any properties that have been set on an object*

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_fileset]
set_property top_file {C:/Data/sources/fftTop.v} [current_fileset]
```

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_fileset 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
```

This example is the same as the prior example, except that it uses the **-dict** option to set all the properties on the synthesis run in a single **set_property** command:

```
set_property -dict [ list flow {Vivado Synthesis 2013} \
STEPS.SYNTH_DESIGN.ARGS.FANOUT_LIMIT 500 \
STEPS.SYNTH_DESIGN.ARGS.GATED_CLOCK_CONVERSION on STEPS.SYNTH_DESIGN.ARGS.FSM_EXTRACTION one_hot ] [get_runs
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 and Temperature Grade.

Syntax

`set_speed_grade` [**`-temperature`** *`arg`*] [**`-quiet`**] [**`-verbose`**] [*`value`*]

Returns

String result

Usage

Name	Description
[**-temperature**]	Temperature grade used for timing analysis (Not available for 7 Series and earlier)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*value*]	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

-temperature - (Optional) Specify the temperature grade for UltraScale devices, used for timing analysis.

NOTE: This option is not available for 7 series devices, or earlier

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 [-toggle_rate arg] [-default_toggle_rate arg]
[-static_probability arg] [-default_static_probability arg]
[-signal_rate arg] [-hier] [-quiet] [-verbose] [objects...]
```

Returns

Nothing

Usage

Name	Description
[**-toggle_rate**]	Toggle rate (%) is the rate at which the output of synchronous logic element switches compared to a given clock input. It is modeled as a percentage between 0 - 200%. A toggle rate of 100% means that on average the output toggles once during every clock cycle, changing on both rising and falling clock edges, and making the effective output signal frequency equal to the clock frequency. Default: 0.0
[**-default_toggle_rate**]	The default toggle rate to be used in power analysis on the primary inputs of the design. The default toggle rate is set on those primary input nets whose switching activity is not specified by the user, simulation data or constraints of the design. Valid values are: 0 <= value < 200. The default value is 12.5. Default: 12.5
[**-static_probability**]	Static probability value: 0 <= Value <= 1 Default: 0.5
[**-default_static_probability**]	The default static probability to be used in power analysis on the design. The default static probability is set on those primary inputs whose switching activity is not specified by the user, simulation data or constraints of the design. Valid values are: 0 <= Value <= 1. The default value is 0.5. Default: 0.5
[**-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 *objects* option. This option should be used only with *objects* 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 on the current synthesized or implemented design. 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.

The **set_switching_activity** command can be used to specify default activity rates for the whole design, or to define the activity of one or more signals in the design.

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.

NOTE: The **reset_switching_activity** is used to reset switching activity for specified objects. Use the **set_switching_activity -default_toggle_rate** or **-default_static_probability** to change or reset these values

Arguments

-default_toggle_rate *rate* - (Optional) The default toggle rate to be used in power analysis on the primary inputs of the current design. The default toggle rate is set on the primary input nets whose switching activity is not specified by the user, simulation data, or constraints of the design. On asynchronous inputs the toggle rate is set with respect to the highest clock in the design. Valid values are: 0 <= value < 200. The default value is 12.5.

NOTE: This option cannot be specified with *objects* as it specifies the default toggle rate for the entire design

-type *value* - (Optional) The type of logic entity for the defined switching activity. Valid types are: registers, inputs, outputs, inouts, ports, outputEnable, three_states, dsps, brams, bramWrite, bramEnable, clockEnable.

-default_static_probability *value* - (Optional) The default static probability to be used in power analysis on the current design. The default static probability is set on primary inputs whose switching activity is not specified by the user, simulation data, or constraints of the design. Valid values are 0 <= value <= 1. The default value is 0.5.

NOTE: This option cannot be specified with *objects* as it defines the default for the entire design. Use **-static_probability** to define the *value* for specific objects.

-static_probability *value* - (Optional) The static switching probability for the specified *objects* to be used in power analysis. Valid values are 0 <= *value* <= 1. The default value is 0.5.

-signal_rate *value* - (Optional) The signal frequency to be used for analysis, expressed as millions of transitions per second (Mtr/s). Specifies the number of times an element changes state per second, including transitions from low-to-high and from high-to-low. 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 hierarchically to signals in the specified hierarchical *objects*. Without **-hier**, the switching activity is applied to the specified *objects* at the current level of the hierarchy.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, and net objects to which the switching activity constraint should be applied.

Examples

The following example specifies a signal rate and switching probability for all ports, then reports the switching attributes for those ports:

```
set_switching_activity -signal_rate 55 -static_probability .33 [get_ports]
report_switching_activity [get_ports]
```

The following example specifies the default switching probability for the current design:

```
set_switching_activity -default_static_probability .75
```

See Also

- get_clocks
- get_nets
- get_ports
- power_opt_design
- report_power
- report_switching_activity
- reset_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
system_jitter	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 returns nothing if successful, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-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
[**-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 femotovolts (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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
hdl_object	Set the value to the given hdl_object
value	Set the value to the given hdl_object

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
objects	Objects to show Find Results view

Categories

GUIControl

Description

Populates the specified objects into the Find Results window in the Vivado IDE.

NOTE: This command is only useful when run in the Vivado IDE. When run in Tcl or Batch mode the command simply returns without error or comment

Arguments

-name *arg* - The name of the report tab to open in the Find Results window. If no name is specified, the default name of **find_1** is provided.



TIP: *If the command is run multiple times, without the **-name** option, each occurrence of **show_objects** will overwrite the prior results.*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - The design objects to show in the Find Results window. The objects must be specified as design objects, and not just specified by name. Objects can be returned by commands like **get_cells**, **get_pins**, **get_nets**, or by **all_inputs**, and **all_rams**.

Examples

The following example shows all DSP objects in the current design in the Find Results window.

```
show_objects -name All_DSPs [all_dsps]
```

The following example shows all of the cells in the design hierarchy that are Clock or DSP PRIMITIVE_TYPEs:

```
show_objects -name find_1 [get_cells -hierarchical \
-filter { PRIMITIVE_TYPE =~ CLK.*.* || PRIMITIVE_TYPE =~ MULT.dsp.* } ]
```

See Also

- all_inputs
- all_rams
- get_cells
- get_nets
- get_pins
- get_ports

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
objects	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 is only useful when run in the Vivado IDE. When run in Tcl or Batch mode the command simply returns without error or comment

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
ports	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-verbose**]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Simulation

Description

Step the current simulation to the next executable statement in the HDL source files.

The line stepping feature lets you run the simulator stepping through the source code line-by-line. This is helpful if you are interested in observing how each line or feature of your HDL source affects the results of simulation.

The step command returns information related to the next executable line from the HDL source file, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 the current executable line of the HDL source code, and pauses at the next executable line, returning information about that line:

```
step
Stopped at time : 0 fs : File "C:/Data/ug937/sim/testbench.v" Line 17
```

See Also

- run
- stop

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

Categories

Simulation

Description

The **stop** command pauses the current simulation. This command can be used within a condition, defined by **add_condition**, to pause the simulation when the condition is true.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 condition named resetLow, that becomes true when the reset signal is low, and then puts a message to the standard output, and stops the current simulation:

```
add_condition -name resetLow {/testbench/reset == 0 } {
puts "Condition Reset was encountered at [current_time]. Stopping simulation."
stop }
```

See Also

- add_condition
- report_conditions
- restart
- run
- step

stop_gui

Close GUI.

Syntax

```
stop_gui [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-verbose**]	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
hw_sio_scans	hardware SIO scans

Categories

Hardware

Description

Stop the specified scan running in the Vivado serial I/O analyzer.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized Eye Scan hardware of Xilinx UltraScale devices or 7 Series FPGAs. The Vivado serial I/O analyzer feature lets you to create, run, and save link scans.

This command lets you stop a scan that is in progress as started with the **run_hw_sio_scan** command.

You can remove the created scan object using **remove_hw_sio_scan**.

This command returns a message if successful, or returns an error if the command fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_scans - (Required) Specify one or more hw_sio_scan objects to stop. The hw_sio_scan must be specified as an object as returned by the **get_hw_sio_scans** commands.

Example

The following example launches a scan using the run_hw_sio_scan command, and then stops the specified scan:

```
run_hw_sio_scan [lindex [get_hw_sio_scans {SCAN_3}] 0]
stop_hw_sio_scan [get_hw_sio_scans SCAN_3]
```

See Also

- create_hw_sio_scan
- current_hw_device
- get_hw_sio_scans
- remove_hw_sio_scan
- run_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

stop_hw_sio_sweep

Stop hardware SIO sweeps.

Syntax

stop_hw_sio_sweep [**-quiet**] [**-verbose**] *hw_sio_sweeps*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
hw_sio_sweeps	hardware SIO sweeps

Categories

Hardware

Description

Stop the specified sweep scan.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized features of Xilinx UltraScale devices or 7 Series FPGAs. It can also be helpful to run multiple scans on a the link with different configuration settings for the GTs. This can help you determine which settings are best for your design. The Vivado serial I/O analyzer feature enables you to define, run, and save link sweeps, or collections of link scans run across a range of values.

This command lets you stop a sweep scan that is in progress as started with the **run_hw_sio_sweep** command.

You can remove the created sweep scan object using **remove_hw_sio_sweep**.

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

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_sweeps - (Required) Specify one or more hw_sio_sweep objects to stop. The hw_sio_sweep must be specified as an object as returned by the **get_hw_sio_sweeps** command.

Example

The following example stops the specified running sweep scan:

```
stop_hw_sio_sweep [lindex [get_hw_sio_sweeps {SWEEP_0}] 0]
```

See Also

- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_sweeps
- remove_hw_sio_sweep
- run_hw_sio_sweep
- wait_on_hw_sio_sweep
- write_hw_sio_sweep

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
aloc	First location (port/cell/site - should be of same type as 'bloc')
bloc	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

aloc - (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 *aloc* 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]
[-shreg_min_size arg] [-mode arg] [-fsm_extraction arg]
[-keep_equivalent_registers] [-resource_sharing arg]
[-control_set_opt_threshold arg] [-max_bram arg] [-max_dsp 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 *name*=<value> -generic <name>=<value> ...
[**-verilog_define**]	Specify verilog defines. Syntax: -verilog_define *macro_name*[=<macro_text>] -verilog_define *macro_name*[=<macro_text>] ...
[**-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

Name	Description
[**-shreg_min_size**]	Minimum length for chain of registers to be mapped onto SRL Default: 3
[**-mode**]	The design mode. Values: default, out_of_context Default: default
[**-fsm_extraction**]	FSM Extraction Encoding. Values: off, one_hot, sequential, johnson, gray, auto Default: auto
[**-keep_equivalent_registers**]	Prevents registers sourced by the same logic from being merged. (Note that the merging can otherwise be prevented using the synthesis KEEP attribute)
[**-resource_sharing**]	Sharing arithmetic operators. Value: auto, on, off Default: auto
[**-control_set_opt_threshold**]	Threshold for synchronous control set optimization to lower number of control sets. Default: 4
[**-max_bram**]	Maximum number of block RAM allowed in design. (Note -1 means that the tool will choose the max number allowed for the part in question Default: -1
[**-max_dsp**]	Maximum number of block DSP allowed in design. (Note -1 means that the tool will choose the max number allowed for the part in question Default: -1
[**-quiet**]	Ignore command errors
[**-verbose**]	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.

You can also use the **synth_design** command to elaborate RTL source files, and open an elaborated design:

```
synth_design -rtl -name rtl_1
```

This command returns a transcript of the synthesis process, or returns an error if it fails.

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 enable pins when available. This optimization can eliminate logic and simplify the netlist. Refer to the **GATED_CLOCK** property in the *Vivado Design Suite User Guide: Synthesis (UG901)* for more information. Valid values for this option are:

- **off** - Disables the conversion of clock gating logic during synthesis, regardless of the use of the **GATED_CLOCK** property in the RTL design.
- **on** - Converts clock gating logic based on the use of the **GATED_CLOCK** property in the RTL design.
- **auto** - lets Vivado synthesis perform gated clock conversion if either the **GATED_CLOCK** property is present in the RTL, or if the Vivado tool detects a gate with a valid clock constraint.

-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** - Run the default synthesis process.
- **runtimeoptimized** - Perform fewer timing optimizations and eliminate some RTL optimizations to reduce synthesis run time.

-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 on clock nets during synthesis. Specified as a value >= 1, which should not exceed the BUFG count on the target device. The default value is 12.

NOTE: Vivado synthesis infers up to the number of BUFGs specified, including those instantiated in the RTL source. For example, with the default setting of **-bufg 12**, if there are three BUFGs instantiated in the RTL, the tool infers up to nine more for a total of 12.

-no_lc - (Optional) Disable the default LUT combining feature of Vivado synthesis.

-fanout_limit *arg* - (Optional) Specify a target limit for the maximum net fanout applied during synthesis. Specified as a value >= 1, the default value is 10,000. This option is applied by Vivado synthesis when the number of loads exceeds an internal limit, and is only a guideline for the tool, not a strict requirement. When strict fanout control is required for specific signals, use the MAX_FANOUT property instead.

NOTE: **-fanout_limit** does not affect control signals (such as set, reset, clock enable). Use the MAX_FANOUT property to replicate these signals as needed

-shreg_min_size *arg* - (Optional) Specified as an integer, this is the minimum length for a chain of registers to be mapped onto SRL. The default is three.

-mode [default | out_of_context] - (Optional) Out of Context mode specifies the synthesis of an IP module, or block module, for use in an out-of-context design flow. This mode turns off I/O buffer insertion for the module, and marks the module as OOC, to facilitate its use in the design. Refer to the *Vivado Design Suite User Guide: Designing with IP (UG896)* or the *Vivado Design Suite User Guide: Hierarchical Design (UG905)* for more information.

-fsm_extraction *arg* - (Optional) Finite state machine (FSM) encoding is automatic (**auto**) 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.

NOTE: Use **-fsm_extraction off** to disable finite state machine extraction in Vivado synthesis. This will override the FSM_ENCODING property when specified

-keep_equivalent_registers - (Optional) Works like the KEEP property 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 5 registers would be moved to the D input logic, rather than using the reset line of a register. However, if **-control_set_opt_threshold** is set to 4, then the reset line is used. This option is specified as an integer greater than or equal to 1; the default setting is 4.

-max_bram *arg* - (Optional) Specify the maximum number of Block RAM to add to the design during synthesis. Specify a value >= 1, which should not exceed the available BRAM count on the target device. If a value of -1 is used, the Vivado synthesis tool will not exceed the available Block RAM limit of the device. The default value is -1.

NOTE: A value of 0 directs Vivado synthesis to not infer BRAMs in the design, but is not a recommended value

-max_dsp *arg* - (Optional) Specify the maximum number of DSPs to add to the design during synthesis. Specify a value >= 1, which should not exceed the available DSP count on the target device. If a value of -1 is used, the Vivado synthesis tool will not exceed the available limit of the device. The default value is -1.

NOTE: A value of 0 directs Vivado synthesis to not infer DSPs in the design, but is not a recommended value

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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

synth_ip

Generate a synthesis netlist for an IP.

Syntax

`synth_ip` [**`-force`**] [**`-quiet`**] [**`-verbose`**] *`objects`*

Returns

Nothing

Usage

Name	Description
[**-force**]	Force regeneration of the netlist.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	All the objects for which a netlist needs to be generated for.

Categories

Project, IPFlow

Description

This command is used in the non-project flow to create a synthesized design checkpoint file (DCP) to support the out-of-context (OOC) IP flow, or to synthesize and implement an IP module in the OOC hierarchical design flow. IP objects are specified by the **get_ips** command, or for the specified IP core file (XCI) as specified by the **get_files** command.



IMPORTANT: *To enable this functionality, the IP core must be marked for OOC generation by setting the GENERATE_SYNTH_CHECKPOINT property to true (or 1) using the **set_property** command on the XCI file*

For project-based designs you would use the **create_ip_run** and **launch_runs** commands. Refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG898)* for more information on Project and Non-Project Modes in Vivado.

The **synth_ip** command will automatically generate any required target files prior to synthesizing the IP core. The source files required to synthesize the IP are copied into the IP run directory. Upon completion, any newly generated OOC target files (dcp, stub files, funcsim netlists...) are registered with the associated IP core.

Arguments

-force - (Optional) Force re-synthesis of the specified IP objects, even if the generated output products for the specified IP are all current.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 IP objects to synthesize. The objects can be specified as IP cores by the **get_ips** command, or as XCI files using the **get_files** command.

Examples

The following example synthesizes the specified IP object, regenerating the netlist if the synthesized core is up-to-date:

```
synth_ip [get_ips char_fifo] -force
```

See Also

- create_ip_run
- get_files
- get_ips
- launch_runs
- synth_design
- read_checkpoint

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) Tie up or down the unused pins of the specified cell objects.

NOTE: The **-of_objects** option requires objects to be specified using the **get_*** commands, such as **get_cells** or **get_pins**, rather than specifying objects by name. In addition, **-of_objects** cannot be used with a search *pattern*.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 -of_objects [get_cells cpuEngine]
```

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*cells*]	Match engine names against cell names Default: *

Categories

IPIntegrator

Description

This command is intended to undo the grouping of IP Integrator cells into a hierarchical module, by either the **group_bd_cells** or the **move_bd_cells** commands. The command moves the cells inside a selected hierarchical module up one level to the parent cell, and then removes the hierarchical module.

The connections between the selected cells are maintained. The connections between these cells and other cells are maintained automatically by removing unneeded subsystem ports and pins.

This command returns 0 if successful, or an error message if it fails.

Arguments

-prefix *arg* - (Optional) A prefix name to apply to any cells that are moved up one level in the hierarchy.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) A hierarchical module defined by the **group_bd_cells** or **move_bd_cells** commands. Only one hierarchical module can be specified for the command at one time.

Example

The following example :

```
ungroup_bd_cells -prefix up2_ [get_bd_cells myMod2]
```

See Also

- get_bd_cells
- group_bd_cells
- move_bd_cells

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
[**-color_index**]	Color index
[**-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 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) Valid value is an integer from 1 to 19. 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
cell_list	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 [**-force_gui**] [**-fileset** *arg*] [**-quiet**] [**-verbose**]

Returns

Nothing

Usage

Name	Description
[**-force_gui**]	Execute this command, even when run interactively in the GUI.
[**-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

-force_gui - Update the compile order in GUI mode of the Vivado IDE.

-fileset *arg* - Update the compile order of the specified fileset.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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] [-buffer_ports]
[-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.
[**-buffer_ports**]	buffer all the ports of 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 converts the cells to a black box cell.

The **update_design** 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.

update_design -cells <args> [-strict] [-from_file <arg>] [-from_design <arg>] [-from_cell <arg>] [-black_box] [-buffer_ports] [-quiet] [-verbose]

Arguments

-cells *args* - (Required) Defines a list of hierarchical leaf-cell 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.

-buffer_ports - (Optional) Insert buffers for all the ports of a black box cell.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-from_files**]	New files and directories to use for updating
[**-norecurse**]	Recursively search in specified directories
[**-to_files**]	Existing project files and directories to limit updates to
[**-filesets**]	Fileset name
[**-force**]	Overwrite imported files in the project, even if read-only, if possible
[**-report_only**]	Do no actual file updates, but report on updates that otherwise would have been made
[**-quiet**]	Ignore command errors
[**-verbose**]	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 <repo_path_list> [current_fileset]'.

Syntax

```
update_ip_catalog [-rebuild] [-add_ip arg] [-delete_ip arg]
[-delete_mult_ip args] [-create_index] [-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
[**-create_index**]	Cache the specified repository's data on disk, to improve load time.
[**-repo_path**]	Used in conjunction with rebuild, add_ip, delete_ip, delete_mult_ip or create_index 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.

-delete_mult_ip *arg* - (Optional) Remove the specified IP cores from the IP repository. This argument requires the **-repo_path** argument to also be specified. The IPs are specified either as paths to the `component.xml` files, or as the VLNV properties of the IP. The VLNV property refers to the Vendor:Library:Name:Version string which identifies the IP in the catalog.

-create_index - (Optional) Cache the specified repository's data on disk, to improve load time. This argument requires the **-repo_path** argument to also be specified.

-repo_path *arg* - (Optional) Used in conjunction with **-rebuild**, **-add_ip**, **-delete_ip**, **-delete_mult_ip** or **-create_index** to specify the directory name of an IP repository to operate on.

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[**-absolute_grid**]	Use absolute grid for relative locations
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
macro	Macro to update
rlocs	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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:

```
{cell0 XmYn cell1 XmYn ... cellN 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_mem

Update Memory.

Syntax

```
update_mem -meminfo arg [-data arg] -bit arg [-proc arg] -out arg
[-force] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
-meminfo	Design mem info file
[**-data**]	Input elf file
-bit	Input bitstream file
[**-proc**]	Instance path of the processor
-out	Output bitfile
[**-force**]	Overwrite existing output file
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

UpdateMem

update_timing

Update timing.

Syntax

```
update_timing [-full] [-skip_delay_calc] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[**-full**]	Perform a full timing update instead of an incremental one
[**-skip_delay_calc**]	Skip delay calculation
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Timing

upgrade_bd_cells

Upgrade configurable IPIntegrator cell(s) to later version.

Syntax

upgrade_bd_cells [**-latest** *arg*] [**-quiet**] [**-verbose**] *objects*...

Returns

List of IPIntegrator cell names those were upgraded, "" if failed

Usage

Name	Description
[**-latest**]	Upgrade the IPIntegrator block to the latest version
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
objects	IPIntegrator cells to be upgraded

Categories

IPIntegrator

Description

Upgrade IP Integrator cells to the latest version available in the IP Integrator catalog.

This command lets you update IP Integrator subsystem designs from an earlier release to use the IP cores from the latest catalog.

This command returns the list of IP Integrator cells that were upgraded, or returns an error if it fails.

Arguments

-latest - (Optional) Upgrade the specified cells to the latest available version from the IP catalog. This option is the default, and is not required.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 - The IP Integrator cells to upgrade to the latest version. These objects must be specified using the **get_bd_cells** command. Objects cannot be referenced by name.

Example

The following example upgrades the specified cells to the latest version available in the IP Integrator catalog:

```
upgrade_bd_cells [get_bd_cells {vidOut1 cmpy_1 newMod1}]
INFO: [BD 41-1162] The cell '/vidOut1' is already at its latest version.
INFO: [BD 41-1162] The cell '/cmpy_1' is already at its latest version.
WARNING: [BD 41-1082] Hierarchy block (/newMod1) cannot be directly upgraded.
Please dive into the hierarchy and select individual cells to upgrade.
```

NOTE: A warning message is returned for the user-defined hierarchical module

See Also

get_bd_cells

upgrade_ip

Upgrade a configurable IP to a later version.

Syntax

upgrade_ip [**-srcset** *arg*] [**-vlnv** *arg*] [**-quiet**] [**-verbose**] [*objects*...]

Returns

A return code indicating success or failure

Usage

Name	Description
[**-srcset**]	(Optional) Specifies the source file set containing the IP to be upgraded Default: The current source fileset Values: Source set name
[**-vlnv**]	(Optional) Identifies the Catalog IP to which the IP will be upgraded. The VLNV string maps to the IPDEF property on the IP core. This is a strict comparison, and the upgrade will fail if the identified IP does not exist in the Catalog. Default: Latest version of the current IP Values: A string of the form '*vendor*:<library>:<name>:<ver sion>'
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*objects*]	IP to be upgraded Values: IP instance(s) within the design, as returned by 'get_ips ' or 'get_bd_cells <cell name>'

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 IP that explicitly supports upgrading. The UPGRADE_VERSIONS property on the ipdef object indicates if there are upgrade versions for an IP core.



TIP: *The upgrade_ip command also accepts Block Design cell IP instances by name, bd_cell objects, or file location. The command upgrades the bd_cell objects within the Block Design, and does not require the diagram to be open in the Vivado IP Integrator feature*

Arguments

-srcset *arg* - (Optional) Specifies the source file set to upgrade the IP files in. If not specified, the default source file set is `sources_1`.

-vlnv *arg* - (Optional) Specify the Vendor:Library:Name:Version attribute of the IP to upgrade from the IP catalog. The VLNV attribute identifies the object in the IP catalog.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 IP cores of Block Design cells to upgrade.

Examples

The following example upgrades all IP cores in the current project to the latest version:

```
upgrade_ip
```

See Also

- create_ip
- get_bd_cells
- import_ip
- open_bd_design

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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*hw_ilas*]	List of hardware ILA objects. Default: Current hardware ILA

Categories

Hardware

Description

Upload the captured data from the memory buffers of the specified ILA debug cores on the Xilinx FPGA hardware device, and move it into a hw_ila_data object in the Vivado logic analyzer.

You can upload captured data from the ILA debug core at any time during the capture process triggered by the **run_hw_ila** command. However, you may want to use the **wait_on_hw_ila** command in any Tcl scripts, to wait until the sample data buffers of the ILA core are fully populated with data. If you run the **upload_hw_ila_data** command prior to this, you may see a message as follows:

```
INFO: [Labtools 27-1965] The ILA core 'hw_ila_1' trigger was stopped by user \
at 2014-Mar-06 08:59:30
INFO: [Labtools 27-2212] The ILA core 'hw_ila_1' captured '6' windows with \
'64' samples each, and a last partial window with '0' samples.
```

The upload process creates a hw_ila_data object in the process of moving the captured data from the ILA debug core, hw_ila, on the physical FPGA device, hw_device. The hw_ila_data object is named after the hw_ila core it is uploaded from.



TIP: *Each hw_ila object has only one matching hw_ila_data object associated with it. Each time upload_hw_ila_data is run for a specific hw_ila core, the hw_ila_data object is overwritten if it already exists*

The data object, hw_ila_data can be viewed in the waveform viewer of the Vivado logic analyzer by using the **display_hw_ila_data** command, and can be written to disk using the **write_hw_ila_data** command.

This command returns a hw_ila_data object, or returns an error if it fails.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_ilas - (Optional) Specify one or more hw_ila objects to upload data from. The hw_ila objects can either be specified as objects returned by the **get_hw_ilas** or **current_hw_ila** commands, or specified by name. If the hw_ila is not specified, the data will be uploaded from the **current_hw_ila**.

Example

The following example arms the current hardware ILA debug core on the target hw_device, captures sample data at the probes as trigger events or capture conditions are encountered. Tcl script processing is suspended while sample data is captured, and then the data is uploaded from the hw_ila on the hw_device, into a hw_ila_data object:

```
run_hw_ila -trigger_now [current_hw_ila]
wait_on_hw_ila [current_hw_ila]
upload_hw_ila_data [current_hw_ila]
```

See Also

- current_hw_device
- current_hw_ila
- current_hw_ila_data
- display_hw_ila_data
- get_hw_devices
- get_hw_ilas
- get_hw_ila_datas
- run_hw_ila
- wait_on_hw_ila
- write_hw_ila_data

validate_bd_design

Run Parameter Propagation for specified design or for a specific cell.

Syntax

`validate_bd_design` [**`-design`** *`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
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

IPIntegrator

Description

Validate an IP Integrator subsystem design, or IP cell or hierarchical module.

This command returns success, and TCL_OK if successful, or TCL_ERROR if it fails, unless **-quiet** is specified.

Arguments

-design *arg* - (Optional) The IP Integrator subsystem design to validate. If not specified, the current IP Integrator subsystem design is validated.

-cell *arg* - (Optional) The IP Integrator cell or hierarchical module to validate. If not specified, the whole design, starting at the top-level is validated.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 validates the IP Integrator subsystem design, and returns a few warnings and errors related to the design:

```
validate_bd_design
INFO: [IP_Flow 19-3236] Customization errors found. Restoring to previous valid
configuration.
ERROR: [IP_Flow 19-508] Validation failed for parameter 'Write_Depth_A'. Value
'268435456' is out of the range (2,9011200)
ERROR: [Common 17-39] 'set_property' failed due to earlier errors
ERROR: [BD 41-66] Error running post_propagate TCL procedure: ERROR: [Common 17-39]
'set_property' failed due to earlier errors.
    while executing
"rdi::set_property CONFIG.WRITE_DEPTH_A 268435456
/microblaze_1_local_memory/lmb_bram"
    invoked from within
"set_property CONFIG.WRITE_DEPTH_A $mem_depth_a $ip"
    (procedure "::xilinx.com_ip_blk_mem_gen_8.0::post_propagate" line 49)
    invoked from within
"::xilinx.com_ip_blk_mem_gen_8.0::post_propagate /microblaze_1_local_memory/lmb_bram
 {}"
WARNING: [BD 41-680] Ranges of mapped paired address blocks
</microblaze_1_local_memory/ilmb_bram_if_cntlr/SLMB/Mem> and
</microblaze_1_local_memory/dlmb_bram_if_cntlr/SLMB/Mem> do not match.
ERROR: [BD 41-241] Message from IP propagation TCL of
/microblaze_1_local_memory/lmb_bram: set_property error: Validation failed for
parameter 'Write_Depth_A'. Value '268435456' is out of the range (2,9011200)
INFO: [xilinx.com:ip:axi_intc:3.1-1] /microblaze_1_axi_intc: The AXI INTC core has
 been configured to operate with synchronous clocks.
ERROR: [Common 17-39] 'validate_bd_design' failed due to earlier errors.
```

See Also

- create_bd_design
- open_bd_design
- save_bd_design

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
[*ips*]	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
[*hw_ilas*]	hardware ILA objects. Default: Current hardware ILA

Categories

Hardware

Description

Suspend Tcl script or Tcl command processing until the ILA debug core memory is filled by captured data samples.

This command is used after the **run_hw_ila** command to pause Tcl processing to wait for the data buffers to fill up. When the **wait_on_hw_ila** command returns, the Tcl command or script processing can continue.

With the ILA debug core memory filed with sample data, when Tcl processing resumes, you can upload the captured data samples into an ILA debug core data object, or hw_ila_data object. Use the **upload_ila_data** command to perform this action.

This command operates silently, returning nothing if successful, or returning an error if it fails.

Arguments

-timeout *arg* - (Optional) Wait for this period of time for all data on the ILA debug cores to be captured. If the timeout interval is reached, the wait command is terminated.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_ilas - (Optional) Specify one or more hw_ila objects to wait on. The hw_ila objects must be specified as an object as returned by the **get_hw_ilas** or **current_hw_ila** commands. If the hardware ILA is not specified, the **current_hw_ila** will be run.

Example

The following example waits for all data on the current hardware ILA debug core to be captured:

```
run_hw_ila hw_ila_1 -trigger_now 1
INFO: [Labtools 27-1964] The ILA core 'hw_ila_1' trigger was armed at 2014-Mar-02 13:20:30
wait_on_hw_ila hw_ila_1
display_hw_ila_data [upload_hw_ila_data hw_ila_1]
INFO: [Labtools 27-1966] The ILA core 'hw_ila_1' triggered at 2014-Mar-02 13:20:31
```

See Also

- current_hw_device
- current_hw_ila
- current_hw_ila_data
- display_hw_ila_data
- get_hw_devices
- get_hw_ilas
- run_hw_ila
- upload_hw_ila_data

wait_on_hw_sio_scan

Wait until hardware SIO 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
hw_sio_scans	List of hardware SIO scan objects.

Categories

Hardware

Description

Suspend a Tcl script or Tcl command processing until the specified serial I/O analyzer scan is complete.

This command is used after the **run_hw_sio_scan** command to pause Tcl processing to wait for the scan to complete. When the **wait_on_sio_scan** command returns, the Tcl command or script processing can continue.

This command operates silently, returning nothing if successful, or returning an error if it fails.

Arguments

-timeout *arg* - (Optional) Wait for this period of time for the serial I/O analyzer scan to complete. If the timeout interval is reached, the wait command is terminated.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_scans - (Required) Specify one or more hw_sio_scan objects to wait on. The hw_sio_scan must be specified as an object as returned by the **create_hw_sio_scan** or **get_hw_sio_scans** commands.

Example

The following example waits for the serial I/O analyzer scan to complete:

```
run_hw_sio_scan [lindex [get_hw_sio_scans {SCAN_0}] 0]
wait_on_hw_sio_scan [lindex [get_hw_sio_scans {SCAN_0}] 0]
```

See Also

- create_hw_sio_scan
- current_hw_device
- get_hw_sio_scans
- remove_hw_sio_scan
- run_hw_sio_scan
- stop_hw_sio_scan
- write_hw_sio_scan

wait_on_hw_sio_sweep

Wait until hardware SIO sweep has completed.

Syntax

```
wait_on_hw_sio_sweep [-timeout arg] [-quiet]
[-verbose] hw_sio_sweeps ...
```

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
hw_sio_sweeps	List of hardware SIO sweep objects.

Categories

Hardware

Description

Suspend a Tcl script or Tcl command processing until the serial I/O analyzer sweep scan is complete.

This command is used after the **run_hw_sio_sweep** command to pause Tcl processing to wait for the sweep scan to complete. When the **wait_on_sio_sweep** command returns, the Tcl command or script processing can continue.

This command operates silently, returning nothing if successful, or returning an error if it fails.

Arguments

-timeout *arg* - (Optional) Wait for this period of time for the serial I/O analyzer sweep scan to complete. If the timeout interval is reached, the wait command is terminated.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

hw_sio_sweeps - (Required) Specify one or more hw_sio_sweep objects to wait on. The hw_sio_sweep must be specified as an object as returned by the **create_hw_sio_sweep** or **get_hw_sio_sweeps** commands.

Example

The following example launches an SIO sweep scan and waits for the sweep to complete:

```
run_hw_sio_sweep [lindex [get_hw_sio_sweeps {SWEEP_0}] 0]
wait_on_hw_sio_sweep [lindex [get_hw_sio_sweeps {SWEEP_0}] 0]
```

See Also

- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_sweeps
- remove_hw_sio_sweep
- run_hw_sio_sweep
- stop_hw_sio_sweep
- write_hw_sio_sweep

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
run	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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` **`[-force]`** **`[-no_mig_contents]`** **`[-no_ip_version]`** **`[-quiet]`** **`[-verbose]`** *name*

Returns

TCL_OK, TCL_ERROR if failed

Usage

Name	Description
[**-force**]	Flag to overwrite existing file.
[**-no_mig_contents**]	Flag to not include MIG PRJ contents into generated Tcl script, but instead will load PRJ from working directory. Default is to include MIG PRJ contents in Tcl script.
[**-no_ip_version**]	Flag to not include the IP version as part of the IP VLNV in create_bd_cell commands. NOTE - this may have implications if there are major IP version changes.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
name	Name exported Tcl file

Categories

IPIntegrator

Description

Export the current IP Integrator subsystem design as a Tcl script file to the disk.



IMPORTANT: *Any directory in the path specified by the name option must already exist, or the script will not be created*

The Tcl script file lets you recreate, reuse, and customize IP Integrator subsystem designs without having to archive the original subsystem design.

When working with a new software release, you must use the output script from the **write_bd_tcl** command to create a block design in the same software release as the Tcl script was generated. This ensures the availability of the needed versions of any IP used in the script. You can then migrate the created block design into a new software release.

This command returns TCL_OK if successful, or TCL_ERROR if it fails, unless **-quiet** is specified.

Arguments

-no_mig_contents - (Optional) Do not include MIG PRJ contents into the generated Tcl script. By default, MIG PRJ contents will be included in the Tcl script.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Tcl 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.

Example

The following example creates a Tcl script from the current IP Integrator subsystem design:

```
write_bd_tcl C:/Data/myDesign.tcl
```

See Also

- create_bd_design
- open_bd_design
- save_bd_design

write_bitstream

Write a bitstream for the current design.

Syntax

```
write_bitstream [-force] [-verbose] [-raw_bitfile] [-no_binary_bitfile]
[-mask_file] [-readback_file] [-logic_location_file] [-bin_file]
[-reference_bitfile arg] [-quiet] file
```

Returns

Nothing

Usage

Name	Description
[**-force**]	Overwrite existing file
[**-verbose**]	Print write_bitstream options
[**-raw_bitfile**]	Write raw bit file (.rbt)
[**-no_binary_bitfile**]	Do not write binary bit file (.bit)
[**-mask_file**]	Write mask file (.msk)
[**-readback_file**]	Write readback files (.rbd, .msd)
[**-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
file	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.

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`.

-readback_file - (Optional) Lets you perform the Readback function by creating the necessary readback files (.rbd, .msd).

- .rbd - An ASCII file that contains only expected readback data, including pad words and frames. No commands are included.
- .msd - An ASCII file that contains only mask information for verification, including pad words and frames. No commands are included.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
file	Design bmm file Values: A filename with alphanumeric characters and .bmm extension.

Categories

FileIO

write_bsdl

Generate a design specific post-configuration BSDL file (.bsd).

Syntax

`write_bsdl [-force] [-bsd` *`arg`*`] [-quiet] [-verbose]` *`file`*

Returns

Name of the output file

Usage

Name	Description
[**-force**]	Overwrite existing .bsd file
[**-bsd**]	Specify an updated generic BSDL file.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Output file name. The .bsd extension is optional.

Categories

FileIO

Description

Generate a Boundary Scan Description Language (BSDL) file (.bsd) for the current design that reflects the post-configuration boundary scan architecture of the target device.

The boundary scan architecture for the device is changed when the device is configured because certain connections between the boundary scan registers and pad may change. These changes must be communicated to the boundary scan tester through a post-configuration BSDL file. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on the available configuration modes.

The **write_bsdl** command reads a pre-configuration BSDL file for the target part from the Vivado Design Suite installation area, and combines that with post-configuration data from the current design.

This command returns the name of the output BSDL file, or returns an error if it fails.

Arguments

-force - (Optional) Overwrite an existing BSDL file of the same name.

-bsd *arg* - (Optional) Specify an existing BSDL file to update. Use this to update a generic BSDL file from the Vivado Design Suite installation with post-configuration data from the current design.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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) Specify the output file name. The `.bsd` extension is optional.

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.

Example

The following example writes a BSDL file at the specified location:

```
write_bsdl -force C:/Data/project/design1.bsd
```

See Also

write_bitstream

write_cfgmem

Create file(s) for programming flash memory.

Syntax

```
write_cfgmem [-force] -format arg -size arg [-interface arg]
[-checksum] [-loadbit arg] [-loaddata arg] [-quiet] [-verbose] file
```

Returns

Nothing

Usage

Name	Description
[**-force**]	Overwrite existing file
-format	Format of the file to generate
-size	Size of memory that is being targeted in M Bytes (must be power of 2).
[**-interface**]	Interface used to program device. Default: SMAPx8
[**-checksum**]	Calculate a 32-bit checksum for each file. Memory will be filed with value of 0xFF unless a different byte value is specified. Default: 0xFF
[**-loadbit**]	Load bit files into memory from given address.
[**-loaddata**]	Load data into memory from given address.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	The name of the file to generate

Categories

FileIO

Description

Formats a design specific configuration bitstream (.bit) file, and any specified data files, into a specified memory configuration file format to program into a flash memory device using the **program_hw_cfgmem** command.

The process whereby the design specific data is loaded or programmed into the Xilinx® FPGA is called configuration. The **create_hw_cfgmem** command defines a flash memory device used for configuring and booting the hardware device.

After the hw_cfgmem object is created, and associated with the hw_device, the configuration memory can be programmed with the bitstream and other data from a memory configuration file created with the **write_cfgmem** command. The hw_cfgmem object is programmed using the **program_hw_cfgmem** command.

The **write_cfgmem -loadbit** command loads one or more specified bitstream files into the memory configuration file, filling the available memory of the device in an upward or downward direction from a specified starting address. You can also add data files to the memory configuration file, by specifying the starting address to load the file with **-loaddata**.



TIP: *When using **-loadbit** and **-loaddata** to fill the memory of the device, you must exercise care to insure that the bitstream and data files fit into the available memory and do not overwrite each other. Any data collisions will cause the **write_cfgmem** command to fail with an error*

The write_cfgmem command returns a transcript of its process when successful, or returns an error if it fails.

Arguments

-force - (Optional) Overwrite a file of the same name if one exists.

-format [BIN | HEX | MCS] - (Required) The format of the memory configuration file to write. Supported values include BIN, HEX, and MCS.

-size *arg* - (Required) Specify the size limit in MBytes of the PROM device that is being targeted. The size must be specified as a power of 2.

-interface *arg* - (Optional) Specify the interface used to program the PROM device. Valid values include SMAPx8 (default), SMAPx16, SMAPx32, SERIALx1, SPIx1, SPIx2, SPIx4, SPIx8, BPIx8, BPIx16. This also determines if byte swapping is enabled or disabled. The default interface is SMAPx8.



IMPORTANT: *The specified interface format of the configuration memory file is critical to properly programming the flash memory device with the **program_hw_cfgmem** command. You should be careful to use this option to match the generated file with the target cfgmem_part*

-checksum - (Optional) Calculate a 32-bit checksum for the PROM file. The device memory will be filled with the default value of 0xFF unless a different byte value is specified. This option generates a checksum value appearing in the memory configuration file. This value should match the checksum in the device programmer. Use this option to verify that correct data was programmed into the flash memory.

-loadbit *arg* - (Optional) Specify the starting address of the PROM device to begin loading one or more bitstream files. The option is specified as a string with the form:

```
"up|down 0x0 bitfile.bit <bitfile2.bit>"
```

Where:

- **up | down** - This option loads one or more BIT files into memory, starting from the specified address, in either and upward or a downward direction.
- **0x0** - The staring address to load the bitstream, specified as a hexadecimal value.
- *bitfile.bit* - The bitstream (.bit) file to load into the flash memory device. You can specify multiple bitfiles, causing the files to be concatenated in a daisy chain.



TIP: *You can only specify the **-loadbit** option once, but you can repeat the arguments as needed to load multiple bitstream files from different starting addresses:*

```
-loadBit "up 0 bitfile1.bit up 0xFFFFFF bitfile2.bit"
```

-loaddata *arg* - (Optional) Load the specified data files into the memory of the configuration device from the starting address. The **-loaddata** option is a string in the same form as the **-loadbit** argument, specifying the direction, starting address, and data file names to add into the memory configuration file. Data files will be added to the flash memory device as is, with no additional formatting.

- **up | down** - This option loads one or more DATA files into memory, starting from the specified address, in either and upward or a downward direction.
- **0x0** - The staring address to load the data file, specified as a hexadecimal value.
- *data_file* - A data file to load into the flash memory device. You can specify multiple data files, causing the files to be concatenated in a daisy chain.

NOTE: Although both **-loadbit** and **-loaddata** are marked as optional, at least one argument must be used to provide the data for the memory configuration file

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 memory configuration file to write. The file extension will match the format specified (`.mcs`), and is not required as part of the file name.

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.

Example

The following example writes the specified memory configuration file in the MCS format, with a size limit of 64 MB, loading the specified bitstream file moving up from the starting address:

```
write_cfgmem -format MCS -size 64 -loadbit "up 0x0 \
C:/Data/Vivado_Debug/project_debug/project_debug.runs/impl_1/sinegen_demo.bit" \
config_memory1
```

See Also

- create_hw_cfgmem
- current_hw_device
- delete_hw_cfgmem
- get_cfgmem_parts
- get_property
- program_hw_cfgmem
- readback_hw_cfgmem
- set_property
- write_bitstream

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
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_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
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
file	Debug probes file name (default extension is .ltx)

Categories

FileIO, Debug

Description

Writes a Vivado Design Suite logic analyzer probes file containing ILA debug cores and signal probes added to the current design. The debug probes data file typically has a `.ltx` file extension.

ILA cores are added to the design using the **create_debug_core** command. ILA probes are added to the design using the **create_debug_port** command, and connected to nets in your design using the **connect_debug_port** command.

The specific information and use of the debug probes file is described in the *Vivado Design Suite User Guide: Vivado Programming and Debugging (UG908)*.

Arguments

-force - (Optional) Overwrite the specified file if it already exists.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
file	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 **-pblocks** or **-cell** option 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 **-pblocks** and **-cell** options 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
file	hardware ILA data file name
[*hw_ila_data*]	hardware ILA data object Default: Current hardware ILA data

Categories

Hardware

Description

Write the ILA debug core sample data, stored in the hw_ila_data object, to a binary file on the disk.

A hw_ila_data object is created by the **upload_hw_ila_data** command when moving the captured data from the physical FPGA device, hw_device.

The **write_hw_ila_data** lets you write the data of the hw_ila_data object to a binary file on the disk for later review. You can read the ILA debug core data back into the Vivado logic analyzer using the **read_hw_ila_data** command, which creates a new hw_ila_data object.

This command returns the name of the file written, or returns an error if it fails.

Arguments

-force - (Optional) Overwrite an existing file of the same name if one exists.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 ILA data file to write. The default file extension for an ILA data file is `.ila`.

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.

hw_ila_data - (Optional) The hardware ILA data to write to the specified file. The hw_ila_data must be specified as an object, as returned by the **get_hw_ila_datas** or the **current_hw_ila_data** commands. If no hw_ila_data object is specified, the current hw_ila_data is written to the specified file.

Example

The following example uploads the data from the hw_ila debug core into a hw_ila_data object, and then writes that data object to the specified ILA data file, overwriting an existing file if one exists:

```
write_hw_ila_data -force design1_ila_data [upload_hw_ila_data hw_ila_1]
```

See Also

- current_hw_ila
- current_hw_ila_data
- get_hw_ilas
- get_hw_ila_datas
- run_hw_ila
- wait_on_hw_ila

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
file	hardware SIO_scan file name
hw_sio_scan	hardware SIO scan data object

Categories

Hardware

Description

Write the populated hw_sio_scan object after run_hw_sio_scan completes.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized Eye Scan hardware of Xilinx UltraScale devices or 7 Series FPGAs. The Vivado serial I/O analyzer feature lets you to create, run, and save link scans.

This command saves the scan to disk after completing the scan run. The format of the file is a CSV file of values observed while running the scan.

This command returns the filename of the file output, or returns an error if the command fails.

Arguments

-force - (Optional) Overwrite the specified file if it already exists.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 scan file to write. The default suffix of `.csv` will be assigned to the scan file if a suffix is not 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.

hw_sio_scan - (Required) Specify a hw_sio_scan object to write to disk. The hw_sio_scan must be specified as an object as returned by the **get_hw_sio_scans** command.

Example

The following example writes the specified hw_sio_scan object to disk, over writing a file of the same name if one exists:

```
write_hw_sio_scan -force C:/Data/Vivado_Debug/LoopBack_1.csv \
[get_hw_sio_scans {SCAN_3}]
```

See Also

- create_hw_sio_scan
- create_hw_sio_sweep
- get_hw_sio_scans
- get_hw_sio_sweeps
- run_hw_sio_scan
- run_hw_sio_sweep
- remove_hw_sio_scan
- stop_hw_sio_scan
- wait_on_hw_sio_scan
- write_hw_sio_scan

write_hw_sio_sweep

Write sweep data to a directory.

Syntax

`write_hw_sio_sweep` **`[-force]`** **`[-quiet]`** **`[-verbose]`** *`directory hw_sio_sweep`*

Returns

Name of the output directory

Usage

Name	Description
[**-force**]	Overwrite existing directory
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
directory	hardware SIO_sweep directory path
hw_sio_sweep	hardware SIO sweep data object

Categories

Hardware

Description

Write the populated hw_sio_sweep object after run_hw_sio_sweep completes.

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized features of Xilinx UltraScale devices or 7 Series FPGAs. It can also be helpful to run multiple scans on a the link with different configuration settings for the GTs. This can help you determine which settings are best for your design. The Vivado serial I/O analyzer feature enables you to define, run, and save link sweeps, or collections of link scans run across a range of values.

This command saves the specified link sweep object to disk after it has been populated by the run_hw_sio_sweep command.

This command returns the name of the directory created, or returns an error if the command fails.

Arguments

-force - (Optional) Overwrite the specified directory if it already exists.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

directory - (Required) The name of a directory to write multiple scan files as a result of the **run_hw_sio_sweep** command.

hw_sio_sweep - (Required) Specify a hw_sio_sweep object to write to disk. The hw_sio_sweep must be specified as an object as returned by the **get_hw_sio_sweeps** command.

Example

The following example writes the specified hw_sio_sweep object to disk:

```
write_hw_sio_sweep sweep_results [get_hw_sio_sweeps {SWEEP_1}]
```

See Also

- create_hw_sio_scan
- create_hw_sio_sweep
- current_hw_device
- get_hw_sio_scans
- get_hw_sio_sweeps
- remove_hw_sio_sweep
- run_hw_sio_sweep
- stop_hw_sio_sweep
- wait_on_hw_sio_sweep
- write_hw_sio_sweep

write_hwdef

Writes hardware definition for use in the software development.

Syntax

```
write_hwdef -[-force] arg -file arg [-quiet] [-verbose]
```

Returns

Sucess/failure status of applied action

Usage

Name	Description
-[-force]	Overwrites the existing hardware definition file
-file	Hardware definition file (Values: A filename with alphanumeric characters and .hwdef extention.)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Project

write_ibis

Write IBIS models for current floorplan.

Syntax

```
write_ibis [-force] [-allmodels] [-nopin] [-truncate arg]
[-component_name 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.
[**-nopin**]	Disable inclusion of the per-pin modeling of the package (path from the die pad to the package pin). Package is reduced to a single RLC transmission line model applied to all pins and defined in the [Package] section. Default: This option is not set. IBISWriter includes per-pin modeling of the package as RLC matrices in the [Define Package Model] section if this data is available.
[**-truncate**]	Maximum length for a signal name in the output file. Names longer than this will be truncated. This property can be set to truncate signal name length to 20, 40, or 0 (unlimited). Default: Truncate signal name length to 40 characters in accordance with the IBIS version 4.2 specification. Default: 40
[**-component_name**]	Specify a new component name for use in multiple FPGA designs to replace the default.
[**-ibs**]	Specify an updated generic IBIS models file.
[**-pkg**]	Specify an updated per pin parasitic package data file.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Output file name. The .ibs extension is optional.

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

-force - (Optional) Overwrite the IBIS file if it already exists.

-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.

-component_name *arg* - (Optional) Specify a new component name to change the default value, which is the device family.

-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 tool installation hierarchy under the parts directory. This argument is required for any parts that do not have generic models in the installation directory.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 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.

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_inferred_xdc

Write file with inferred xdc timing constraints.

Syntax

```
write_inferred_xdc [-force] [-all] [-append] [-async_clocks]
[-clock_groups] [-clocks] [-excl_clocks] [-exceptions]
[-io_constraints] [-merge_existing_constraints] [-name arg] [-quiet]
[-verbose] [file]
```

Returns

Nothing

Usage

Name	Description
[**-force**]	Overwrite existing file.
[**-all**]	Generate all constraints except missing clocks which are generated with the -clocks option
[**-append**]	Append the constraints to file, don't overwrite the constraints file
[**-async_clocks**]	Find asynchronous clock groups
[**-clock_groups**]	Find asynchronous and exclusive clock groups, equivalent to options -async_clocks -excl_clocks
[**-clocks**]	Find missing clock definitions
[**-excl_clocks**]	Find logically and physically exclusive clock groups
[**-exceptions**]	Find missing exceptions
[**-io_constraints**]	Find missing input and output delays
[**-merge_existing_constraints**]	Add existing user defined constraints to the generated constraints
[**-name**]	Start constraints wizard in a GUI panel with this name. Do other command options can be combined with -name.
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
[*file*]	Filename to write constraints into

Categories

FileIO

write_mem_info

Write design mem info to an xml file.

Syntax

`write_mem_info [-force] [-quiet] [-verbose]` *`file`*

Returns

The name of the xml file

Usage

Name	Description
[**-force**]	Overwrite existing mem info xml file
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Design mem info file Values: A filename with alphanumeric characters and .mmi extension.

Categories

FileIO

write_peripheral

Save peripheral component to the disk.

Syntax

`write_peripheral` `[`**`-quiet`**`] [`**`-verbose`**`]` *`peripheral`*

Returns

Nothing

Usage

Name	Description
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
peripheral	Peripheral object

Categories

Project, IPFlow, CreatePeripheral

Description

Write the specified AXI peripheral object to disk in the form of the `component.xml` file. The peripheral is written to the repository location specified by the **create_peripheral** command, under the name specified at creation.

Arguments

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

peripheral - (Required) The peripheral object to write. The peripheral is created with the **create_peripheral** command, and should be captured in a Tcl variable to facilitate further processing by this and other related commands. See the example below.

Example

This example creates a new AXI peripheral, with the VLNV attribute as specified, and captures the peripheral object in a Tcl variable for later processing, then adds AXI slave interfaces to the peripheral, and generates the output products for the peripheral, and writes the component.xml file to disk. The directory of the new peripheral is added to the IP_REPO_PATH property of the current fileset, and the IP catalog is updated to include the new peripheral:

```
set perifObj [ create_peripheral {myCompany.com} {user} {testAXI1} {1.3} \
-dir {C:/Data/new_periph} ]
add_peripheral_interface {S0_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
add_peripheral_interface {S1_AXI} -interface_mode {slave} -axi_type {lite} $perifObj
generate_peripheral -driver -bfm_example_design -enable_interrupt $perifObj
write_peripheral $perifObj
set_property ip_repo_paths  C:/Data/new_periph [current_fileset]
update_ip_catalog -rebuild
```

See Also

- add_peripheral_interface
- create_peripheral
- generate_peripheral

write_project_tcl

(User-written application).

Syntax

```
write_project_tcl [-paths_relative_to arg] [-target_proj_dir arg]
[-force] [-all_properties] [-no_copy_sources] [-absolute_path]
[-dump_project_info] [-quiet] [-verbose] file
```

Returns

True (0) if success, false (1) otherwise

Usage

Name	Description
[**-paths_relative_to**]	Override the reference directory variable for source file relative paths Default: Script output directory path
[**-target_proj_dir**]	Directory where the project needs to be restored Default: Current project directory path
[**-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
[**-absolute_path**]	Make all file paths absolute wrt the original project directory Default: 0
[**-dump_project_info**]	Write object values Default: 0
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Name of the tcl script file to generate

Categories

xilinxtclstore, projutils, user-written

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

-target_proj_dir - (Optional) Specify the directory path where the project will be recreated. The tool will write the **create_project** command with the directory path specified with this option.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 example exports a Tcl script named "recreate.tcl" for the current project in the "./script" directory and specifies the "/tmp/test" directory for the "create_project" command. When the "recreate.tcl" script is run in the Vivado Tcl shell, the project will be re-created in "/tmp/test" directory:

```
write_project_tcl -target_proj_dir "/tmp/test" ./script/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_schematic

Export schematic.

Syntax

```
write_schematic [-force] [-format arg] [-orientation arg] [-scope arg]
[-name arg] [-quiet] [-verbose] file
```

Returns

Name of the output file

Usage

Name	Description
[**-force**]	Overwrite existing file
[**-format**]	Values: native or pdf. read_schematic can be used to view native format. Default: native
[**-orientation**]	Values: landscape or portrait
[**-scope**]	Values: current_page, visible or all Default: current_page
[**-name**]	Schematic window title
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
file	Output file

Categories

FileIO

Description

Output the currently opened, or specified Schematic window in the Vivado IDE to a file.

The file can be written as a native ASCII file that can be read back into the Vivado IDE using the read_schematic command, or can be written as a PDF file for use outside of the Vivado Design Suite. This can be useful when documenting IP cores from the IP Packager flow, or from the Vivado IP Integrator.

Arguments

-force - (Optional) Overwrite the specified Schematic file if it already exists.

-format [native | pdf] - (Optional) Write the schematic in an ASCII file format, native to the Vivado Design Suite, that can be read back into the tool using the **read_schematic** command, or write the schematic as a PDF for viewing and use outside the Vivado Design Suite. The default is **native**.

NOTE: The **-format** argument is case sensitive

-orientation [landscape | portrait] - (Optional) Write the schematic in a vertical (portrait) or horizontal (landscape) orientation. The default is **portrait**.

-scope [current_page | visible | all] - (Optional) Write the current page of the schematic, or the visible content displayed in the schematic window, or all of the schematic. The default is **current_page**.

-name *arg* - (Optional) Specifies the name of the open Schematic window to write. Use this option to write the specified schematic window when more than one schematic window is opened.

NOTE: When multiple schematic windows are opened, the first opened window is written if **-name** is not specified.

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 Schematic file to write. The path is optional, but if one is not provided the Schematic file will be written to the current working directory, or the directory from which the Vivado tool was launched.

Example

The following example writes the specified Schematic window, displayed in the Vivado IDE, to a native format file that can later be read back into the Vivado Design Suite, overwriting the specified file if it already exists:

```
write_schematic -name Schematic C:/Data/mySchematic.txt -force
```

The following example writes a PDF file with a horizontal view of the specified schematic window:

```
write_schematic -format pdf -name "Schematic (2)" C:/Data/mySchematic.pdf \
-orientation landscape
```

See Also

read_schematic

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
file	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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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_sysdef

Writes hardware definition for use in the software development.

Syntax

```
write_sysdef -[-force] arg -hwdef arg -bitfile arg [-meminfo arg]
-file arg [-quiet] [-verbose]
```

Returns

Sucess/failure status of applied action

Usage

Name	Description
-[-force]	Overwrites the existing hardware definition file
-hwdef	Input Hardware definition file
-bitfile	Bitstream file
[**-meminfo**]	BRAM file
-file	System definition file (Values: A filename with alphanumeric characters and .sysdef extention.)
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution

Categories

Project

write_verilog

Export the current netlist in Verilog format.

Syntax

```
write_verilog [-cell arg] [-mode arg] [-lib] [-port_diff_buffers]
[-write_all_overrides] [-keep_vcc_gnd] [-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, pin_planning, synth_stub, 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
[**-keep_vcc_gnd**]	Don't replace VCC/GND instances by literal constants on load terminals. For simulation modes only.
[**-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: *file*.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
file	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.
- pin_planning - Output only the I/O ports for the top-level of the design.
- synth_stub - Output the ports from the top-level of the design for use as a synthesis stub.
- 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: The **-library** option can only be used for simulation. Vivado synthesis will treat all Verilog files as being in the default work 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 pin_planning** or **-mode synth_stub** 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.

-keep_vcc_gnd - (Optional) By default, when writing a nelist for simulation, or from an IP Integrator block design, the Vivado Design Suite replaces VCC and GND primitives, and the nets they drive, with literal constants on each of the loads on the net. The **-keep_vcc_gnd** option disables this default behavior and preserves the VCC or GND primitives.

-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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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`

In the following example, the functional simulation mode is specified, the option to keep VCC and GND primitives in the output simulation netlist is enabled, and the output file is specified:

```
write_verilog -mode funcsim -keep_vcc_gnd out.v
```

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] [-keep_vcc_gnd] [-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, pin_planning, synth_stub 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
[**-keep_vcc_gnd**]	Don't replace VCC/GND instances by literal constants on load terminals. For simulation modes only.
[**-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
file	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.
- pin_planning - Output only the I/O ports in the entity declaration for the top module.
- synth_stub - Output the ports from the top-level of the design for use as a synthesis stub.

-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 pin_planning** or **-mode synth_stub** 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.

-keep_vcc_gnd - (Optional) By default, when writing a nelist for simulation, or from an IP Integrator block design, the Vivado Design Suite replaces VCC and GND primitives, and the nets they drive, with literal constants on each of the loads on the net. The **-keep_vcc_gnd** option disables this default behavior and preserves the VCC or GND primitives.

-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.

-force - (Optional) Overwrite the VHDL 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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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 `.vhdl` 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] [-add_netlist_placement]
[-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.
[**-add_netlist_placement**]	Export netlist placement 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.

The **write_xdc** command lets you write invalid XDC constraints so that you can quickly report constraints that have been ignored by the Vivado Design Suite due to a problem with the way the constraint is written or applied. This is useful for debugging constraint files applied in specific designs.

This command can be used to create an XDC file from a design with UCF files. All constraints from the active constraint fileset will be exported to the XDC, even if they come from multiple files.

NOTE: The **write_xdc** command will not convert all UCF constraints into XDC format, and is not intended to automatically convert UCF based designs to XDC. Refer to the *Vivado Design Suite Migration Methodology Guide (UG911)* for more information on migrating UCF constraints to XDC

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, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose - (Optional) 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
```

This example writes only the invalid constraints, including both fixed and unfixed cells, to the specified file:

```
write_xdc -constraints invalid C:/Data/bad_constraints.xdc
```

See Also

read_xdc

xsim

Load a simulation snapshot for simulation and return a simulation object.

Syntax

```
xsim [-maxdeltaid arg] [-nolog] [-maxlogsize arg] [-onfinish arg]
[-onerror arg] [-tclbatch arg] [-t arg] [-runall] [-R]
[-testplusarg args] [-vcdfile arg] [-vcdunit arg] [-view args]
[-wdb arg] [-tp] [-tl] [-nosignalhandlers] [-noieeewarnings] [-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)
[**-maxlogsize**]	Set the maximum size a log file can reach in MB. The default setting is unlimited Default: -1
[**-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 may be repeated to open multiple files.
[**-wdb**]	Specify the waveform database output file

Name	Description
[**-tp**]	Enable printing of hierarchical names of process being executed
[**-tl**]	Enable printing of file name and line number of statements being executed.
[**-nosignalhandlers**]	Run with no signal handlers to avoid conflict with security software
[**-noieeewarnings**]	Disable warnings from VHDL IEEE functions
[**-quiet**]	Ignore command errors
[**-verbose**]	Suspend message limits during command execution
snapshot	The name of design snapshot

Categories

Simulation

Description

The **xsim** command loads a simulation snapshot to run a batch mode simulation, or to provide a GUI and/or Tcl-based interactive simulation environment. The snapshot must be generated using the **xelab** command.

Arguments

-maxdeltaid *arg* - (Optional) Specify the maximum number of delta cycles as an integer greater than 0. The default value is 10000.The simulator will report an error if it exceeds the specified maximum number of simulation loops, or delta cycles, at simulation run time. Refer to the *Vivado Design Suite User Guide: Logic Simulation (UG900)* for more information on delta cycles.

-onfinish [stop | quit] - (Optional) Specify the actions of the simulator upon finishing the simulation run. Valid values are stop the simulation, or quit the simulator. The default is stop.

-onerror [stop | quit] - (Optional} Specify the actions of the simulator upon encountering an error. Valid values are stop the simulation, or quit the simulator. The default is stop.

-tclbatch | -t - (Optional) Specify a TCL script file to run the simulator in batch mode.

-runall | -R - (Optional) Run the simulation until no event is left in the event queue, a breakpoint or valid condition is encountered, or a run time exception occurs. Then quit the simulator. This is the equivalent of 'run -all; exit'.

-testplusarg *args* - (Optional) Specify plusargs to be used by **$test$plusargs** and **$value$plusargs** system functions. These arguments are visually distinguished from other simulator arguments by starting with the plus ('+') character.

-vcdfile *arg* - (Optional) Specify a Value Change Dump (VCD) file to capture simulation output.

-vcdunit *arg* - (Optional) Specify the time unit for the VCD output. The default unit is the same as the simulation engine precision.

-view *arg* - (Optional) Open a wave configuration file to store the waveform activity for the simulation. The Wave Config file contains just the list of wave objects (signals, dividers, groups, virtual buses) to display, and their display properties, plus markers. A wave configuration file is written in the current simulation with the **save_wave_config** command.

NOTE: This option may be repeated to open multiple Wave Config files

-wdb *arg* - (Optional) Specify the simulation waveform database (WDB) file. When the simulation completes, the simulation is written to a static simulator database file. The file can be opened for later review by the **open_wave_database** command.

-tp - (Optional) Print the hierarchical names of process being executed to the standard output.

-tl - (Optional) Print the file name and line number of statements being executed to the standard output.

-nosignalhandlers - (Optional) Disables the installation of OS-level signal handlers in the simulation. With the signal handlers disabled, OS-level fatal errors could crash the simulation abruptly with little indication of the nature of the failure.



IMPORTANT: *Use this option only if your security software prevents the simulator from running successfully.*

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

NOTE: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

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

NOTE: Message limits can be defined with the **set_msg_config** command.

snapshot - (Required) The name of the simulation snapshot to be executed. The snapshot must have been previously compiled by xelab.

Examples

The following example launches xsim on the specified simulation snapshot:

```
xsim C:/Data/project_xsim/project_xsim.sim/sim_1/behav/testbench_behav
```

See Also

launch_xsim

XILINX
ALL PROGRAMMABLE™

Appendix A

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 2014.2 Documentation:

http://www.xilinx.com/cgi-bin/docs/rdoc?v=2014.2;t=vivado+docs

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