Verification Continuum™

# HAPS® Prototyping
# Command Reference

April 2022

SYNOPSYS®

solvnetplus.synopsys.com

# Copyright Notice and Proprietary Information



# Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

# Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

# Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

# Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at
http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

# Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 East Middlefield Road
Mountain View, CA 94043
www.synopsys.com

April 2022

# Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

# Contents

## Chapter 3: Partition Constraint File Tcl Commands

## Chapter 4: Target System Specification Commands

## Chapter 5: UPF Commands



## Chapter 6: FDC Constraint Command Reference

## Chapter 7: Netlist Edit Command Reference



## Chapter 8: Tcl Find and Expand

## Chapter 9: Commands for Unified Compile

SYNOPSYS®
Silicon to Software™

CHAPTER 1

# Startup Commands

This chapter describes the command for starting HAPS® ProtoCompiler or ProtoCompiler DX from a Unix command prompt and the individual runtime command options available from either a Unix or a Windows environment:

- ProtoCompiler Startup Commands, on page 12
  Commands to launch the prototyping tool: `protocompiler`, `protocompiler_dx`, and `protocompiler_s` (Unix only)
- protocompiler_runtime Startup Commands, on page 15
  Commands to start the runtime executable for debugging: `protocompiler_runtime`, `protocompiler_dx_runtime`, and `protocompiler_s_runtime` (Unix and Windows)

# ProtoCompiler Startup Commands

The following commands start the ProtoCompiler or ProtoCompiler DX software. Make sure to set the path to *installDirectory*/bin and then enter the appropriate command at the Unix prompt:

```
protocompiler

protocompiler_dx

protocompiler_s
```

---

**Note:** The ProtoCompiler, ProtoCompiler DX, and ProtoCompiler S tools only run on a Unix platform.

---

Starts the FPGA synthesis tool and runs synthesis from the command line. Use the appropriate command for the tool you are using. The command to start the synthesis tool from the command line includes a number of command line options.

## Syntax

**protocompiler**|**protocompiler_dx**|**protocompiler_s** [*options* **...** ]

## Command-Line Options

Command-line options control tool action on startup and, in many cases, can be combined on the same command line.

**-batch**
Starts the synthesis tool in batch mode from the specified Tcl file without opening the tool window.**-evalhostid**
Reports host ID for node-locked and floating licenses.

**-help**
Lists available command line options and descriptions.

**-history** *logFilename*
Records all Tcl commands and writes them to the specified history log file when the command exits.

**-identify_dir** *pathName*

Specifies the location of the installation directory for launching the instrumentor/debugger tool set.**-ip_license_wait** *waitTime*
Specifies how long to wait for a Synopsys DesignWare IP license when one is not immediately available. If you do not specify the -ip_license_wait option, license queuing is not enabled.

If all requested licenses are checked out or if the specified wait time elapses, the tool excludes the IP and continues to process the rest of the design. Any IP block without a license is treated either as an error or a black box.

License queuing allows you to wait until a license becomes available or specify a wait time in seconds. You can use this option in conjunction with the -batch keyword. For details, see *Queuing Licenses*, on page 604 in the *User Guide*.

The *waitTime* value determines license queuing and sets a maximum wait time:

- Undefined or 0 = Queuing off
- 1 = Queuing enabled, indefinite wait time
- >1 = Queuing enabled for the specified time

**-license_release**

Releases FPGA synthesis licenses for a session after the place-and-route job is launched. The software allows place and route to continue running even after exiting the synthesis tool so that it does not consume a license.

This command option must be run in batch mode. Specify one of the following commands:

```
protocompiler -batch -license_release

protocompiler_dx -batch -license_release

protocompiler_s -batch -license_release
```

For details, see Releasing the Tool License During Place and Route, on page 607 in the *User Guide*.

**-licensetype** *featureName*

Specifies a license if you work in an environment with multiple Synopsys FPGA licenses. You can use this option in conjunction with the -batch keyword.

If you have licenses for multiple products, separate each feature license by a colon so that licenses can be searched in the order they are read until an available license is found.

**-license_feature** [*list*]
Specifies a colon-separated list of additional licenses to check out to enable specific features.

**-license_wait** [*time*]
Sets length of time to wait for a license to become available.**-log** *filename*
Writes all output to the specified log file.**-loga** *filename*
Appends all output to the specified log file.

**-max_parallel_jobs** *number*
Sets the maximum number of concurrent processes to use when performing synthesis.

**-nopopup**
Suppresses popup dialog boxes.

**-shell**
Starts synthesis tool in shell mode.

**-tcl** *Tclscript*
Starts the synthesis tool in the graphical user interface using the specified project or Tcl file.

**-tclcmd** *command*
Specifies Tcl command to be executed on startup.

**-verbose_log**
Writes messages to stdout.log in verbose mode.

**-version**
Reports version of the specified synthesis tool.

# protocompiler_runtime Startup Commands

Runtime commands launch the debugger and start support utilities. Separate commands are provided for the ProtoCompiler and ProtoCompiler DX products with additional command options included with the ProtoCompiler runtime command.



## protocompiler_runtime Command

Accesses the ProtoCompiler runtime utilities/executables through a wrapper script made up of the following commands.

**Syntax** **protocompiler_runtime debug** [**-in** *exportLocation*] [**-shell**] [**-tcl** *tclScript*]

**protocompiler_runtime board_bringup** [**-shell**] [**-tcl** *tclScript*]**debug** [**-in** *exportLocation*] [**-shell**] [**-tcl** *tclScript*]

Opens a debugger session:

- Including the -in option opens a debugger window from the exported runtime location. Exporting the runtime environment creates the directory proto_haps_top/debug in *exportLocation*. The project file is located in the debug directory and is named debug.prj.
- Including the -shell option opens the debugger in shell mode.
- Including the -tcl option opens the debugger from the specified Tcl script.

**board_bringup** [**-shell**] [**-tcl** *tclScript*]

Opens the board bring-up utility:

- Including the -shell option opens the board bring-up utility in shell mode.
- Including the -tcl option opens the board bring-up utility from the specified Tcl script.

## protocompiler_dx_runtime Command

Accesses the ProtoCompiler DX runtime utilities/executables through a wrapper script made up of the command below.

**Syntax** **protocompiler_dx_runtime debug** [**-in** *exportLocation*] [**-shell**] [**-tcl** *tclScript*]**debug** [**-in** *exportLocation*] [**-shell**] [**-tcl** *tclScript*]

Opens a debugger session:

- Including the -in option opens a debugger window from the exported runtime location. Exporting the runtime environment creates the directory proto_haps_dx_top/debug in *exportLocation*. The project file is located in the debug directory and is named debug.prj.
- Including the -shell option opens the debugger in shell mode.

Including the -tcl option opens the debugger from the specified Tcl script.

## protocompiler_s_runtime Command

Accesses the ProtoCompiler runtime utilities/executables through a wrapper script made up of the following commands.

**Syntax** **protocompiler_s_runtime debug** [**-in** *exportLocation*] [**-shell**] [**-tcl** *tclScript*]

**protocompiler_s_runtime board_bringup** [**-shell**] [**-tcl** *tclScript*]**debug** [**-in** *exportLocation*] [**-shell**] [**-tcl** *tclScript*]

Opens a debugger session:

- Including the -in option opens a debugger window from the exported runtime location. Exporting the runtime environment creates the directory proto_haps_top/debug in *exportLocation*. The project file is located in the debug directory and is named debug.prj.
- Including the -shell option opens the debugger in shell mode.
- Including the -tcl option opens the debugger from the specified Tcl script.

**board_bringup** [**-shell**] [**-tcl** *tclScript*]

Opens the board bring-up utility:

- Including the -shell option opens the board bring-up utility in shell mode.
- Including the -tcl option opens the board bring-up utility from the specified Tcl script.

SYNOPSYS®
Silicon to Software™

CHAPTER 2

# Shell Command Reference

This chapter describes shell commands for both the ProtoCompiler and ProtoCompiler DX products.

| | | |
|---|---|---|
| analyst | cdpl_queue | conspeed_hstdm |
| dh_module_sources | dm_root | database |
| design | design_intent | dh_module_sources |
| dm_root | edit | export |
| formal_verify | generate<br>import_vivado_ip | help |
| history | job | launch |
| message_override | option | process_bd_ip |
| report constraint_check | report ctc | report<br>debug_essential_signals |
| report debug_visibility | report<br>external_tool_versions | report instrumentation |
| report list | report messages | report_message_summary |
| report out_of_date | report rtl_diagnostics | report slr_assignment |
| report state_diff | report syntax_check | report target_system |
| report timing | report vc_static | rerun |
| run compile | run debug_eco | run map |

| | | |
|---|---|---|
| run par_explorer | run partition | run postpar_resynthesis |
| run pre_instrument | run pre_map | run pre_partition |
| run system_generate | run system_route | view |

# analyst

Changes and manipulates the schematic views of the netlist.

| | | |
|---|---|---|
| analyst clone_view | analyst flatten | analyst push |
| analyst critical_path | analyst get_selected | analyst select |
| analyst dissolve | analyst group | analyst unfilter |
| analyst filter | analyst pop | analyst view |

## analyst clone_view

Opens a copy of the current view.

**analyst clone_view** *designID*

*designID*
The design ID to clone. If not specified, clones the current view.

## analyst critical_path

Filters the view to show the instances that are part of the critical path, if available.

**analyst critical_path**

## analyst dissolve

Removes targeted hierarchies from the design. Contents of the hierarchy are put into the level that originally contained the hierarchy.

**analyst dissolve** *collection*

*collection*
Collection of instances to dissolve.

## analyst filter

Filters the view by selected instances and ports.

**analyst filter**

## analyst flatten

Flattens the current view.

**analyst flatten**

## analyst get_selected

Returns the names of currently selected objects.

**analyst get_selected** [-inst] [-net] [-pin] [-port]

**-inst**
Returns the names of selected instances. If no -*type* option is set, the names of all selected objects are returned.

**-net**
Returns the names of selected nets. If no -*type* option is set, the names of all selected objects are returned.

**-pin**
Returns the names of selected pins. If no -*type* option is set, the names of all selected objects are returned.

**-port**
Returns the names of selected ports. If no -*type* option is set, the names of all selected objects are returned.

## analyst group

Creates a graphical group of instances.

**analyst group** [*collection*] [**-name** *name*]

*collection*
Instances to group. All instances must be on the same level of the hierarchy.

**-name** *name*
Specifies a name for the created graphical group.

## analyst pop

Pops up the hierarchy.

**analyst pop**

## analyst push

Pushes down the hierarchy.

**analyst push** *hierarchyName*

*hierarchyName*
The name of the group or instance to push the hierarchy down into.

## analyst select

Selects specified objects.

**analyst select** [*collection*] [**-append**] [**-clear**] [**-instances**][**-primitives**]

*collection*
The ID of the collection to select.

**-append**
Appends objects to the selection list.

**-clear**
Clears the selection list.

**-instances**
Selects all instances in the current view.

**-primitives**
Select all primitive instances in the current view.

## analyst unfilter

Unfilters the view.

**analyst unfilter**

## analyst view

Opens a schematic view.

**analyst view** *designID*

*designID*
The design ID to view.

# cdpl_queue

Use the cdpl_queue command to set a CDPL queue, get the configuration setting of the CDPL queue, or clear the CDPL queue settings. These settings are valid only for the current session, and not saved in the .ini file or any other script. Create a Tcl file to save the settings and use it for subsequent sessions of the ProtoCompiler tool. This command does not have a GUI equivalent.

## Syntax

**cdpl_queue -set | -get | -clear** [*queueType*][*queueConfig*]

The cdpl_queue command includes the following options:

**-set** [*queueType*][*queueConfig*]

Sets a CDPL queue configuration.

- queueType is lowmem, highmem, mediummem, default or vendor. The queue type vendor is for all place and route jobs.
- queueConfig is the queue host file. See CDPL help documentation on how to set up a queue host file.

Example:

```
cdpl_queue -set highmem /remote/sbg_ui/myqueue/myhighmem.sge
```

**-get** [*queueType*]

Returns the configuration setting of the queueType specified. If queueType is not given, the command returns all the queue types set in the current session.

**-clear**

Clears all the queue settings.

To use this command, add the following string in your script file in the ProtoCompiler tool:

```
set PWD <working directory path>
set env(CDPL_LOGDIR) $PWD/cdpllogs
```

```
option set cdpl 1
cdpl_queue -set default $PWD/hosts_default
cdpl_queue -set vendor $PWD/hosts_vendor
cdpl_queue -set lowmem $PWD/hosts_lowmem
cdpl_queue -set mediummem $PWD/hosts_mediummem
cdpl_queue -set highmem $PWD/hosts_highmem
```

- hosts_default - To set up host files.
- cdpllogs - From the CDPL_LOG directory, access the log files from a typical CDPL setup. In the cdpllogs directory, check the files master.XXX.XXX.XXX.log and worker.XX.XX.XXX.log to verify how the cdpl_queue command executes during the synthesis flow.

For more information, refer to the CDPL (Common Distributed Processing Library) documentation.

## CDPL Configuration Queue Type

cdpl_queue command supports five queue types. Specific queue types are used for specific primary jobs.

- vendor - Used for running place and route jobs.
- highmem - Used for high memory consumption jobs such as the partition, global optimization, pre-partition, or the GCC flow in distributed synthesis mode.
- mediummem - Used for medium memory consumption jobs such as distributed synthesis running in QOR mode.
- lowmem - Used for low memory consumption jobs such as area estimation or distributed compiler/fast distributed synthesis.
- default - Used if queueType is not given and the current session is used to run the job. This queue type is used for all other jobs.

## Memory Requirement Recommendations

| | |
|---|---|
| vendor | 64GB + 8core for Xilinx devices |
| highmem | • Partition flow, global opt/pre-partition - 256GB or more depending on the design.<br>• Synthesis flow, GCC/advanced synthesis in distributed synthesis - 32GB or more depending on the design. |
| mediummem | Distributed synthesis in QOR mode - 16GB or more depending on the design |
| lowmem | Area estimation, distributed compiler/fast distributed synthesis - 8 GB or more depending on the design. |
| default | • Partition flow - 256GB<br>• Synthesis flow -<br>64GB - if place and route job is combined<br>32GB - if place and route is not combined |

# conspeed_hstdm

Generates a test design that verifies HT3 cable connections and HSTDM channel connectivity for the specified TSS.

Design files are copied to a directory named conspeed_hstdm in your current working directory.

## Syntax

**conspeed_hstdm**
**-bitrate 800|1000|1100|1200|1400**
**-clear**
**-flow implementation|runtime**
**-hstdm_ratio**
**-init**
**-mode unidir|bidir**
**-rundir** *dirPath*
**-serial_number** *number* | **-multihost** *multihostList.txt*
**-tdm_type HSTDM|HSTDM_ERD|DIRECT**
**-technology HAPS70|HAPS80|HAPS80D**
**-tss** *TSSfile*

**-bitrate 800|1000|1100|1200|1400**
Sets the HSTDM bitrate. The default is 1400.

**-clear**
Cleans the old build for rerun. Recommended before rerun.

**-flow implementation|runtime**
Sets the flow value. implementation generates a bit file. runtime executes runtime. The default is implementation.

**-fpga_run serial|parallel|CDPL**
Runs a SLP project in serial, parallel or distributive mode. Default is parallel.

**-hstdm_ratio**
MUX ratio used in the design. The defaults are 8 for HSTDM and 7 for HSTDM_ERD.

**-init**
Copies the necessary file from installation location to directory specified in -rundir. This is a mandatory step before implementation flow.

**-mode unidir|bidir**
Sets the mode. Values are unidir (one direction per cable), or bidir (both directions per cable, each direction driving half of the cable traces). Default is bidir.

-**multihost** *multihostFile.txt*
Specifies a multi-host file that includes multiple chains (superchains) in CSV format. See To specify multiple hosts in the following table for an example. This option is valid only during the runtime flow. For single systems use the **-serial_number** option.

**-rundir** *dirPath*
Specifies the directory where the test design will be synthesized. The default is ./conspeed_hstdm.

**-serial_number** {*number*}
Specifies hardware serial number to validate design on a HAPS system. Serial number is used to get the UMRBus address. *number* must be inside curly braces. A single serial number is required for a single HAPS system. For superchains use the **-multihost** option.

**-tdm_type HSTDM|HSTDM_ERD|DIRECT**
Specifies the type of TDM used. Default is **HSTDM.**

**-technology HAPS70|HAPS80|HAPS80D**
Sets the HAPS system. For mixed systems (HAPS-70 and HAPS-80), use the newer technology. The default is HAPS80. (HAPS-80D cannot be in a mixed system)

**-tss** *TSSfile*
Sets the TSS file to use. The default is board.tcl in the current directory.

## Examples

| To ... | Do ... |
|---|---|
| Copy necessary files from installation. | `conspeed_hstdm -init -rundir` *directoryName* |
| Clean-up before rerunning. | `conspeed_hstdm -clear -rundir` *directoryName* |

| To ... | Do ... |
|---|---|
| Use implementation flow to generate bit file for given user set-up (TSS). | `conspeed_hstdm -bitrate 1100 -mode bidir -technology HAPS80 -flow implementation -tss board.tcl -rundir test` |
| To specify multiple hosts | If the hardware setup uses multiple hosts, where each host has multiple-chain (superchain) setups (for example Host 1 has 2 superchains and Host 2 has 3 superchains) provide a CSV file to the conspeed_hstdm command. In this example, we are creating a CSV file (host_slno.txt) with the following information.<br><br>`Host1, H000001`<br>`Host1, H000002`<br>`Host2, H000003`<br>`Host2, H000004`<br>`Host2, H000005`<br><br>Make sure that the order of the HAPS systems in the csv file match the order in the TSS file. For example, FB1 should be H000001, and FB2 should be H000002, etc.<br><br>`conspeed_hstdm -mode bidir -flow runtime -multihost host_slno.txt -tss myTss.tcl -rundir myTest` |
| Input serial number for validation. | `conspeed_hstdm -serial_number {H123456} -flow runtime -tss myTss.tcl -rundir myTest` |

# dh_module_sources

Returns a list of the source files that define a module hierarchy. You can also use this command to identify files that are not required for the specified module hierarchy.

## Syntax

**dh_module_sources** *moduleName* [**-of**] [**-noself**]

*moduleName*
Specifies the module (view) name of the module hierarchy.

**-of**
Includes list of sources referencing the module.

**-noself**
Excludes sources in which the module definition occurs.

## Examples

```
dh_module_sources work.sub2.verilog
```

```
dh_module_sources -of -noself work.sub2.vhdl
```

## See Also

- *dm_root*, on page 29

# dm_root

The dm_root command displays the name of the top-level module for the current design in batch mode.

## Syntax

**dm_root**

## Example

```
% dm_root
```

For example, this is the result of this command:

```
work.test.verilog
```

## See Also

- *dh_module_sources*, on page 28

# database

Creates and manipulates a repository for the design project and sets the context to execute commands within the ProtoCompiler design flow.

| | | |
|---|---|---|
| apply_state -backannotate | apply_state -idc\|idclist | apply_state -import_est |
| apply_state -import_vivado | apply_state -link_module | apply_state -import_est |
| apply_state -link_module | archive | clone_state |
| close | create | get_state |
| list_state | load | query |
| query_state | remove_state | rename_state |
| set_state | unarchive | unlock_state |

For information about using databases, see Working with a Design Database, on page 58 in the *User Guide*.

## database apply_state -backannotate

Backannotates the imported net-delay values from Vivado for a more accurate post place-and-route timing analysis from a mapped database state. or backannotates the data from running source level timing analysis from the system-generate database state prior to place and route.

**apply_state -backannotate [-bg]**

**-bg**

Starts the compilation process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**apply_state -backannotate gsv**

**gsv**

This command is used to instruct the tool to perform GSV backannotation only, and skip all of the other steps in the backannotation process. This significantly reduces the time taken for backannotation in the debug flow.

## database apply_state -idc|-idclist

The command is used to apply IDC changes incrementally. Use the command to add more debug signals directly to a specific state after the design has gone through an entire flow. In Standard compile flow, the tool recompiles only the required modules affected by the IDC changes. In Unified Compile flow, the tool calls out only the Instrumentor for adding the required changes instead of regenerating the entire pre-partition node.

Supported for compile database states. Requires incremental_debug option set to 1 before running compile state.

**database apply_state -idc | -idclist** *value* [**-overwrite**]

**-idc | -idclist** *value*
Apply debug data from IDC files into current state netlist.

**-overwrite**
Use this argument to overwrite all the current IDC signals. When this option is provided, the already instrumented signals are overwritten by the provided IDC file. When this option is not provided, the new debug signals are appended to the existing signals.

## database apply_state -import_est

Specifies an area estimation file to import. Replaces any existing estimates.

**apply_state -import_est** *file*

## database apply_state -import_vivado

Imports place-and-route results from the specified *sourceDirectory* into a map database state for backannotation or for use at runtime.

**apply_state -import_vivado** *sourceDirectory*

## database apply_state -link_module

Links a single FPGA map database state to an FPGA view in the system-generate database state.

**apply_state -link_module** *moduleName database|mapState*

*moduleName*
The module name to be linked.

*database*|*mapState*
The name of the *database*; *mapState* is a database state within the named database; the *database* and *mapState* arguments are separated by a vertical bar (for example, mydb0 | map0).

## database archive

Archives all or a portion of a database to the specified archive file. It is recommended that you use the GUI version of this command to archive a database. See *Archiving a Database*, on page 67 in the *User Guide* for details.

**archive -name** *directoryPath*
[**-include_envsettings 1**|**0**]
[**-include_source 1**|**0**]
[**-mode full**|**active_tree**|**current**|**lite**]

*directoryPath*
The path to the archive file. The default filename is *databaseName*.sar.

**-include_envsettings 1|0**
Includes/excludes local environment variables in the archive. Saves all environment variable information to file envvariable_settings.txt at the same level as the database.

**-include_source 1|0**
Includes/excludes input source files from the archive. By default, this option is off and all input files are excluded from the archive file.

**-mode full | active_tree | current | lite**
Archives the entire database (full), only the database states from the current state up to root (active_tree), the current state only (current) for creating partial archives for diagnostic use, or the current active tree (lite). The default is active_tree.

Archives of the active tree (-mode active_tree) or the entire database (-mode full) are complete archives, and run commands can be used from these archived database states to generate subsequent states.

With -mode current specified, the archived state cannot be used to generate a subsequent state with a run command. You can only view reports and schematics.

With -mode lite, the created archive contains the minimum data required to recreate the active tree from the root to the current database state. It includes the minimum data to specify run commands from the compiled

state. You must start from the compiled state and specify the run commands sequentially to regenerate subsequent states. For details, see *Creating Minimal (Lite) Archives*, on page 74 in the *User Guide*.

## database clone_state

Creates a copy of a state. This is primarily useful for cloning a system_generate state which can then link to a different set of single-FPGA map states, or for cloning a map state that has multiple Vivado results being explored.

**database clone_state -name** *newCloneName* **[-state** *stateClone***]**

**-name** *newCloneName*
Name of the new cloned state.

**-state** *stateClone*
State to clone rather than the current state.

## database close

Closes the current database design project.

## database create

Creates a new database project. When a database is created, a Tcl variable *$proto*(DB_LOCATION) is set with the path to the directory containing the database.

Use option get valid_technologies for a list of valid technology names or use option get technology to get the technology in use. For example:

```
% option get technology
HAPS-80
```

**create** *directoryPath* [**-default_technology** *value*] [**-technology** *value*]

*directoryPath*
Specify the path to the new database. The argument *directoryPath* is visible through a Tcl variable *$proto*(DB_LOCATION)

**-default_technology** *value*
Specify the default technology expected to be used in the prototype system, for example *HAPS-80*.

**-technology** *value*
Specify the default technology expected to be used in the prototype system. Some examples are HAPS-80, or HAPS-70. Use option get valid_technologies command to see the complete list of valid technology names.

For mixed systems, use the largest system as the technology. For multi-FPGA designs, the technology applies through the compile state. Thereafter, the speed grade and technology are taken from the Target System Specification (TSS). For single-FPGA designs, the technology defined here applies for all subsequent states in this database.

## database get_state

Returns the current database state.

**get_state** [**-next** |**-previous**]

**-next**
Lists the database states following the current state.

**-previous**
Lists the prior database state.

## database list_state

Lists the individual databases in the design project.

**list_state** [**-error_states**] [**-verbose**]

**-error_states**
Lists only states with an error status.

**-verbose**
Lists the database in tree format.

## database load

Loads the root database project from the specified directory path; database load without any arguments loads the database from the active directory if there is only one directory present.

**load** *directoryPath* [**-autocreate**] [**-default_technology** *value*] [**-list**] [**-technology** *value*]

*directoryPath*
Specifies the directory path to the root database. The argument *directory-Path* is visible through a Tcl variable *$proto*(DB_LOCATION).

**-autocreate**
Creates a new database automatically if it does not already exist.

**-default_technology** *value*
Specify the default technology expected to be used in the prototype system, for example *HAPS-80*. Use only when creating a new database with -autocreate.

**-list**
Returns a list of all databases in the current directory.

**-technology** *value*
Specify the default technology expected to be used in the prototype system, for example *HAPS-80*. Use only when creating a new database with -autocreate.

## database query

Returns status information on the current database.

**query** [**-current_schematic**] [**-dir**] [**-name**|**-path**] **[-size] -technology**

**-technology**
Query the technology to be used for run commands from this state.

**-name**
Returns the current database name.

**-path**
Returns the full path to the current database directory.

**-current_schematic**
Returns the name of the database state currently accessible to design query commands; use view schematic to open the design from the current database state.

## database query_state

Returns information on the current database state.

**query_state -status**
**-available_metrics** [*table*.*name*]
**-backannotation**
**-command**
**-dump_metrics** [**-show_queries**] [**-all**] | [**-par**]
**-hdl_define**
**-hdl_param**
**-input_files**
**-is_locked**
**-is_running**
**-link_module**
**-link_state** *moduleID*
**-metric** [*table*.]*name* [**-object** *value*] [**-jobname** *value*]
**-metric_details** [*table*.]*name* [**-object** *value*] [**-jobname** *value*]
**-mode** *activeTree*
**-option** *name*
**-out_of_date**
**-par**
**-run_time**
**-run_type**
**-script** *moduleName*
**-size** *diskSize*
**-state** *databaseState*
**-state_diff** *databaseState*
**-status**
**-technology**
**-top_module**
**-verbose**

**-available_metrics** [[*table*].*name*]

Returns metrics that can be queried for the design in the current state. Shows metrics matching [*table*.]*name* if specified, otherwise shows all metrics in all tables. Return value is a list of lists of the form {{name1 object1 jobname1} {name2 object2 jobname2} ...}.

Examples:

Show a Tcl list of metrics that can be queried for the current state:

```
database query_state -available_metrics
```

A simple loop to show the values of all available metrics:

```
foreach amt [database query_state -available_metrics] { set
metric [lindex $amt 0]; set object [lindex $amt 1]; set job
[lindex $amt 2]; puts "$metric $object: [database query_state
-metric $metric -object $object -jobname $job]" }
```

**-backannotation**
Return timestamp (seconds since epoch) of the most recent backannotation run, or 0 if not run.

**-command**
Returns the current command.

**-dump_metrics** [**-show_queries**] [**-all**]
Returns metrics and values available for the design in its current state. By default show only primary metrics. Use **-all** to show detailed metrics. Use **-show_queries** argument to display Tcl commands to retrieve each metric. The default output format is: <table>.(<object>|global): <metric> = <value> [<units>] from <job> [<description>]
**-show_queries** returns available metrics in the form of commands that can be used to retrieve the metrics.
**-all** returns detailed metrics as well as primary metrics.
Example:
Show human-readable dump of metrics:

```
database query_state -dump_metrics
```

**-hdl_define**
Returns a list of HDL 'defines for current state.

**-hdl_param**
Returns a list of HDL parameters for current state.

**-input_files**
Returns a list of all input files to the current state.

**-is_locked**
Returns 1 if the current state is locked.

**-is_running**
Returns 1 if the process is currently running.

**-link_module**
Lists the module IDs of database links. This argument is only valid when run from a system-generate database state.

**-link_state** *moduleID*
Returns the database state or path linked to the specified module ID. The module ID is the root name used for each individual FPGA database. For example, mb1 is the module ID in these cases: mb1_uA, mb1_uB, mb1_uC, and mb1_uD.

**-metric** *table.name* [**-object** *value*] [**-jobname** *value*]
Queries the value of a QoR metric.
See *Naming Conventions for database query_state Metrics*, on page 38 for details.
*table.name* is the name of the metric to query.
**-object** *value* queries the metric associated with a specific object. If not specified, queries globally.
**-jobname** *value* queries the metric associated with a specific jobname *value*, or any job if not specified.
Examples:
Query the time taken for the compiler run:

```
query_metric runtime.realtime -jobname compiler
```

Set Tcl variable icgr to the number of ICG latches removed from the design:

```
set icgr [query_metric clock_conversion.icg_removed]
```

**-metric_details** [*table.name*] [**-object** *value*] [**-jobname** *value*]
Queries detailed information about a QoR metric.
table.name is the name of the metric to query, proceeded by table. If a table is not specified, the main metric table is assumed.
See *Naming Conventions for database query_state Metrics*, on page 38 for details.
**-object** *value* queries the metric associated with a specific object. If not specified, queries globally.
**-jobname** *value* queries the metric associated with a specific jobname *value*, or any job if not specified.

## Naming Conventions for database query_state Metrics

The naming convention used for metrics consists of the following:

- Table – Represents a group of related metrics, such as, timing, runtime, or clock conversion.
- Metric Name – Descriptive string used to query metrics. This name usually consists of lower case letters with underscores between words.
- Units – Values associated with the metric, such as ns or percent are only shown if details are specified.
- Object – Some metrics are associated with an object, while others are global. Objects can be a clock net name, view name, or an instance path.
- Description – Brief description of the metric.

For example, clock conversion metrics can be specified as follows:

| Table | Name | Description |
|---|---|---|
| clock_conversion | clean_clock_trees | Number of non-gated/non-generated clock trees |
| clock_conversion | clean_clock_pins | Number of clock pins driven by non-gated/non-generated clock trees |
| clock_conversion | gated_clock_trees | Number of gated/generated clock trees |
| clock_conversion | instances_converted | Number of sequential instances converted |
| clock_conversion | instances_notconverted | Number of sequential instances left unconverted |

**-mode** *activeTree*
: For out_of_date and state_diff queries, query just the current state or the entire sequence of states from root. Default is the active tree.

**-option** *name*
: Returns the value of the named option when state was created.

**-out_of_date**
: Returns a Tcl list of out of date states.

**-par**
: Return timestamp (seconds since epoch) of the imported PAR data if PAR was imported, otherwise 0.

**-run_time**
: Returns the time in seconds elapsed for the associated database command to complete.

**-run_type**
: Returns the current run type.

**-script** *moduleName*
: Returns the path to the database generation script for the specified module ID.

**-size** *diskSize*
: Returns on-disk size in bytes.

**-state** *databaseState*
: Addresses queries to the specified database state instead of the current database state.

**-state_diff** *databaseState*
Returns the differences between the named state and current state.

**-status**
Returns the status of the run command (returns 1 for success and 0 for failing).

**-technology**
Query the technology to be used for run commands from this state.

**-top_module**
Returns the name of the top module for the design.

**-verbose**
Show more information about the query.

**-dir**
Returns the directory where database is located.

**-size** *diskSize*
Returns on-disk size in bytes.

## database remove_state

Removes the named database state and all subsequent dependent states within that database hierarchy.

**remove_state** *databaseState*

*databaseState*
The name of the state to be removed.

## database rename_state

Renames a state.

**database rename_state** *dataState* [**-state** *state*]

*dataState*
The data state to be renamed.

**-state** *state*
Renames *state* instead of the current state.

## database set_state

Changes the location to the specified database state.

**set_state** *databaseState*

*databaseState*
Specifies an existing state.

## database unarchive

Extracts the named archive file to the specified database directory. It is recommended that you use the GUI version of this command to unarchive a database. See *Extracting Database Files from an Archive*, on page 71 in the *User Guide* for details.

**unarchive** *archiveFile*.**sar** [**-path** *directoryPath*]

*archiveFile*.**sar**
Specifies the archive file to be extracted.

**-path**
Specifies the database directory where the file is to be extracted; if omitted, the file is extracted to the current database.

## database unlock_state

Unlocks a state. Allows a state to be modified when a tool process terminated abnormally while modifying it.

**unlock_state** *dataState*

*dataState*
Specifies the state to unlock.

# design

*Beta*

Returns netlist data representing information about the design. Commands are available in both batch and GUI mode.

| | | |
|---|---|---|
| design c_diff | design c_symdiff | design get |
| design c_filter | design c_union | design list |
| design c_info | design close | design open |
| design c_intersect | design expand | design set |
| design c_list | design find | design top_level |
| design c_print | design get_prop | |

## design c_diff

Returns a new find collection containing the differences between two existing find collections.

### Syntax

**design c_diff** *collection1 collection2*

*collection1*
The first collection to compare.

*collecton2*
The second collection to compare.

## design c_filter

Filters a find collection based on set properties.

### Syntax

**design c_filter** [**-inst**] [**-net**] [**-pin**] [**-port**] **[-view]** *collection pattern*

*collection*
The collection ID to filter.

*pattern*
Statement used to filter.

**-inst**
Returns matching instances. If no type option (-inst, -net, -port, or -pin) is set, all types are returned.

**-net**
Returns matching nets. If no type option (-inst, -net, -port, or -pin) is set, all types are returned.

**-port**
Returns matching ports. If no type option (-inst, -net, -port, or -pin) is set, all types are returned.

**-pin**
Returns matching pins. If no type option (-inst, -net, -port, or -pin) is set, all types are returned.

**-view**
Returns matching view. If no type option (-inst, -net, -port, or -pin) is set, all types are returned.

## design c_info

Returns information about the contents of a find collection.

### Syntax

**design c_info** [*collection*] [**-array** *value*]

*collection*
Find collection information to display.

**-array** *value*
Specify an array to store collection information in.

## design c_intersect

Defines common objects that are included in each of the collections being compared.

### Syntax

**design c_intersect** *collectionList*

*collectionList*
List of collections separated by spaces.

## design c_list

Converts a collection to a Tcl list of objects.

### Syntax

**design c_list** *collection*

*collection*
Collection to convert.

## design c_print

Displays collections or properties in column format.

### Syntax

**design c_print [-append]** [**-file** *filename*] **[-foot** *object***] [-head** *object***]** [**-prop** *propertyName*] *collection*

**-append**
Appends to the file specified in -file rather than overwriting it.

*collection*
The collection to print as a table.

**-file**
Writes the collection to *filename*.

**-prop**
Writes a column in the table for properties of type *propname*.

**-foot**
Prints footer.

**-head**
Prints header.

## design c_symdiff

Returns a new find collection containing the difference between two existing find collections.

### Syntax

**design c_symdiff** *collection1 collection2*

*collection1* The first collection.

*collection2* The second collection.

## design c_union

Combines multiple collections into a single collection.

### Syntax

**design c_union** *collectionList*

*collectionList*
Space-separated list of collections.

## design close

Closes the specified design ID. If no design ID is provided, this command closes the current active design.

### Syntax

**design close** *designID*

*designID*

The design ID to close.

## design expand

Identifies objects based on their connectivity, by expanding forward from a given starting point. Returns a collection.

### Syntax

**design expand**
[**-from** *object*]
[**-hier**]
[**-leaf**]
[**-level** *integer*]

[**-***objectType*]
[**-print**]
[**-seq**]
[**-thru** *object*]
[**-to** *object*]

**-from** *object*
Specifies a list or collection of ports, instances, pins, or nets for expansion forward from all listed pins. Instances and input pins are automatically expanded to all output pins of the instances. Nets are expanded to all output pins connected to the net. If you do not specify this argument, backward propagation stops at a sequential element.

**-hier**
Modifies the range of any expansion to any level below the current view. The default for the current view is the top level and is defined with the define_current_design command as in the compile-point flow.

**-leaf**
Returns only non-hierarchical instances.

**-level** *integer*
Limits the expansion to N logic levels of propagation. You cannot specify more than one -from, -thru, or -to point when using this option.

**-***objectType*
Optionally specifies the type of object to be returned by the expansion. If you do not specify ***objectType***, all objects are returned. The object type is one of the following:

-inst – returns all instances between the expansion points. This is the default.

-pin – returns all instance pins between the expansion points.

-net – returns all nets between the expansion points.

-port – returns all top-level ports between the expansion points.

**-print**
Evaluates the expand function and prints the first 20 results. If you use this command from HDL Analyst, results are printed to the Tcl window; for constraint-file commands, the results are printed to the log file at the start of the Mapper section. For a full list of objects found, you must use c_print or c_list. Reported object names have prefixes that identify the object type. There are curly braces around each name to allow for spaces in the names. For example:

```
{i:reg1}
{i:reg2}
{i:\weird_name[foo$]}
{i:reg3}
<<found 233 objects. Displaying first 20 objects. Use
c_print or c_list for all. >>
```

**-seq**
Modifies the range of any expansion to include only sequential elements. By default, the expand command returns all object types. If you want just sequential instances, make sure to define the *object_type* with the -inst argument, so that you limit the command to just instances.

**-thru** *object*
Specifies a list or collection of instances, pins, or nets for expansion forward or backward from all listed output pins and input pins respectively. Instances are automatically expanded to all input/output pins of the instances. Nets are expanded to all input/output pins connected to the net. You can have multiple -thru lists for product of sum (POS) operations.

**-to** *object*
Specifies a list or collection of ports, instances, pins, or nets for expansion backward from all the pins listed. Instances and output pins are automatically expanded to all input pins of the instances. Nets are expanded to all input pins connected to the net. If you do not specify this argument, forward propagation stops at a sequential element.

## design find

Identifies design objects based on specified criteria.

### Syntax

**design find**
[**-below** *path*]
[**-depth** *viewNumber*]
[**-filter** *expression*]
[**-flat**]
[**-hier** [**-hsc** *character*]]
[**-inst** *instance*]
[**-net** *net*]
[**-***objectType*] *pattern*

[**-pin** *pin*]
[**-port** *port*]
[**-print**]
[**-seq**]
[**-view** *view*]

**-below** *path*
Sets the start point of the search to the specified instance path. Only searches for objects below this point.

**-depth** *depth*
Sets the start depth for the search. *depth* may be a single hierarchy depth or a range. Using -depth with a range will cause -hier and -flat arguments to be ignored.

**-filter** *expression*
Further refines the results of find by filtering the results using the specified object property. For syntax details, refer to *find -filter*, on page 385.

**-flat**
Extends the search to all levels, but with -flat, the * wildcard character matches hierarchy separators as well as characters. This means that the following example finds instance a1_fft at the current level as well as the hierarchical instance a1.fft:

**-hier**
Extends the search downward through each level of the local hierarchy, instead of limiting the search to the current view. The default hierarchy separator for the search is the period (.).

**-inst** *instance*
Finds instances. If no -type option is set, find defaults to finding instances, nets, and ports.

**-net** *net*
Finds nets. If no -type option is set, find defaults to finding instances, nets, and ports.

**-***objectType pattern*
Specifies the type of object to be found. Object types are view, inst, port, pin, or net. The *pattern* argument is required and specifies the search pattern to be matched. The pattern can include the * and ? wildcard characters (see *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375).

**-pin** *pin*
Finds pins. If no -type option is set, find defaults to finding instances, nets, and ports.

**-port** *port*
Finds ports. If no -type option is set, find defaults to finding instances, nets, and ports.

**-print**
Prints the first 20 search results. For a full list of objects found, use c_print or c_list. If you use find from the shell, the results are printed to the Tcl window; if you find in the constraint file, the results are printed to the log file at the beginning of the Mapper section. Reported object names have prefixes that identify the object type and curly braces around each name to allow for spaces in the names as shown below:

```
{i:reg1}
    {i:\weird_name[foo$]}
    {i:reg2}
    <<found 233 objects. Displaying first 20 objects. Use c_print
    or c_list for all. >>
```

**-seq**
Finds sequential (clocked) instances (the -inst object type is not required). This argument is equivalent to -filter @is_sequential.

**-view** *view*
Finds views. If no -type option is set, find defaults to finding instances, nets, and ports.

find -seq -flat a1*fft

## design get_prop

Returns a list of property values for an object or collection.

### Syntax

**design get_prop** [-all] [*objectName*|*collection*] [**-array** *value*] [**-prop** *value*]

*objectName*|*collection*
The object or collection to use.

**-all**
Print all available properties.

**-array**
Specifies the array to store properties in.

**-prop**
The property value to return.

## design get

Returns the design ID for the current active design.

### Syntax

**design get**

## design list

Returns a list of available design IDs.

### Syntax

**design list**

## design open

View schematic of the design in its current state.

### Syntax

**design open** [**-database** *value*] [**-list**] [*netlist*] [**-verbose**]

**-database** *value*
Path to the database if different than the current database.

**-list**
Lists netlists available for viewing in the current state.

*netlist*
The netlist to view.

**-verbose**
Use with -list. Prints short description of each available netlist.

## design set

Sets specified design ID as the active design.

### Syntax

**design set** *designID*

*designID*

The design ID to set as active.

## design top_level

Returns a Tcl list of top level information. List is ordered: lib, top_module, top_view.

### Syntax

**design top_level**

# design_intent

Manages option settings that control optimization of the design. The command is used to open a Tcl file of options to allow the individual settings to be modified. The customized design-intent file is then saved to a local directory and subsequently called to define the optimization to be applied.

## Syntax

**design_intent**
**-custom** [*template*.**tcl**]
*design_intent*.**tcl**
**-list**

**-custom** [*template*.**tcl**]
Opens a file of options in a text editor for modification. If a template file is not specified, opens the fast_turnaround.tcl file from the installation to use as a template. The customized file is saved to a local directory.

*design_intent*.**tcl**
Uses the specified source Tcl file from the local directory or from either /lib/design_intent/protocompiler or /lib/design_intent/protocompiler_dx in the installation directory to control design optimization.

**-list**
Lists the available design-intent option Tcl files included in the installation.

## Example

To set timing_qor design_intent:

```
design_intent timing_qor.tcl
```

To open fast_turnaround.tcl in the editor for customization:

```
design_intent -custom
```

# edit

A set of commands for editing specific file types. Edit commands create Tcl files for use as runtime arguments and do not alter or change any database states. The commands merely launch the helper application and immediately return the shell prompt.

## Syntax

**edit**
*filename*
**fdc** *fdcFilename* [**-mode** {**gui**|**text**}]
**idc** *idcFilename* [**-mode** {**gui**|**text**}]
**options**
**pcf** *pcfFilename* [**-mode** {**gui**|**text**}]

*filename*
Opens the named file in a text editor window.

**fdc** *fdcFilename* [**-mode** {**gui**|**text**}]
Opens an FDC timing constraint file. When the -mode argument is gui (the default), opens *fdcFilename* in the constraint editor window; when the -mode argument is text, opens the file in a text editor.

**idc** *idcFilename* [**-mode** {**gui**|**text**}]
Opens an instrumentor window or a text editor with the specified IDC file:

When the -mode argument is gui (the default), opens the integrated instrumentor graphical interface using the specified IDC file (the *idcFilename* argument is required, if it does not exist, it is created).

When the -mode argument is text, opens *idcFilename* in a text editor.

**options**
Opens a graphical window that lists the applicable options and their current settings for the selected database state. Values can be changed in the table and saved as a Tcl script.

**pcf** *pcfFilename* [**-mode** {**gui**|**text**}]
Opens a PCF partition constraint file. When the -mode argument is gui (the default), opens *pcfFilename* in the PCF editor window; when the -mode argument is text, opens the file in a text editor.

# export

Makes files accessible outside of the design database. Export commands do not alter or change the state of the design. These are the individual export commands:

| | | |
|---|---|---|
| export file | export options | export_verdi_de |
| export input_file | export report | export verification |
| export idc | export runtime | export vivado |
| export netlist | export srs_bypass | |

## export file

Exports the specified file from the current database state to the indicated directory.

**export file -list** [**-verbose**] [**-all**]

**export file** [**-path** *directoryPath*] *filename*|**-all**

**-list** [**-verbose**] [**-all**]
Lists the available files for export. Shows non-reports by default. Use -all to see more.
If -verbose is included, prints a short description of each file following its name.

**-path** *directoryPath*
Target directory location for file export; default is the current directory.

*filename*|**-all**
The name of the file to export (use export file -list to show available files). The -all argument exports all available files from the current database state.

## export input_file

Export a cached copy of an input file used to generate the current state. This is available for all input files other than input source files since those are not cached in the database. Specify the full path to the file as shown with the **-list** argument, but the leading path to the file will be replaced with the one specified using the **-path** argument.

**-list**
Lists the available files for export.

**-path** *directoryPath*
Target directory location for file export; default is the current directory.

*filename*
The name of the file to export (use export file -list to show available files).

## export idc

Exports the generated IDCs. This command can be used to examine or modify debug signals. Supported in compile and pre_partition database states. Run the export idc command to observe and analyze the generated debug signals after the compile stage is completed. This is the syntax for the command:

**export idc -path** *pathToExportDir*

For example:

```
export idc -path unified_debug_idcs
```

The command exports the following files to the specified directory:

| | |
|---|---|
| sva_identify_compile_out.idc | Written if the RTL contains SVA signals, and SVA signals are enabled in the UTF file. |
| dumpvars_identify_compile_out.idc | Written if the RTL contains $dumpvars specifications, and this instrumentation is enabled. |
| identify_compile_out.idc | Complete IDC file with all the specified debug signals, including RTL instrumentation and signals from IDC specified during compilation. |
| sva_report.txt | Lists SVA information, along with modules and hierarchical instance paths, and RTL source information. Use it t cross-probe the signals to the RTL.<br>Exported only if the appropriate SVA instrumentation is applied. |
| dumpvars_report.txt | Lists the applied signal information along with the modules/hierarchical instance paths, and the RTL source. Use for cross-probing to the RTL.<br>Exported only if the appropriate $dumpvars instrumentation is applied. |

## export netlist

Exports a design netlist from the current state suitable for simulation. Supported for the compile, pre-map, map, pre_partition, partition, system-route, and system-generate states. Multiple netlists may be exportable from each state. From system generate, the default is post-synthesis netlist and is only valid after synthesis of all linked map states, but you can also export the presynth netlist. Use the .vp file generated by run system_generate for prototype simulation with these post-synthesis files. See also *export verification*, on page 60.

You can also use this command from any state to export the setup and generate a script to handle simulation with VCS. Simulation setup does not require that verification_mode option be set to 1. For details about simulating a design, see *Running Simulation*, on page 557 in the *User Guide*.

The export netlist command will write out the required VCS folder (synvcs) in a given directory. For example, when export netlist is done after run compile a directory sim_c0 is created, in which the folder synvcs will be created. All required files, for example, the vcs script (sim_vcs_script.do) and netlists (top_compile.vm) will be placed in the synvcs folder.

**export netlist**
[**-format** *value*]
[-list]
[netlist_name]
[**-path** *targetDirectory*]
[-presynth]
[-verbose]
[-zebu]

**-format** *value*
Export netlist *value* type.

**-list**
Lists all netlists exportable from the current state.

*netlist_name*
Specifies the name of the netlist to export. Use -list to see the available netlists. The default is *default*.

**-path** *targetDirectory*
Specifies the target directory for the exported netlist file; default is the current directory.

**-presynth**
Exports the top-level netlist instead of a single FPGA.

**-verbose**
When used with **-list**, returns a description of each exportable netlist. When used without **-list**, returns nothing.

**-zebu**
Supports the Zebu flow by writing Zebu-compatible netlists from each state.

**Limitations**

- sim_slp_post_synth_script.do does not work when multi-file VM is generated. It is recommended to set syn_multi_file_vm=0 to generate a single VM.
- One module partitioned across different FPGAs will have the same name, which will result in a VCS duplicate module error.

## export options

Exports current option settings as a Tcl file.

**export options** [**-path** *directoryPath*/*filename*] [**-state** *databaseState*]

**-path** *directoryPath*/*filename*
Specifies the full path to the file containing the applicable list of options. The file entries are listed as a line-separated list of options in "option set *name value*" format that can be saved and sourced. The path defaults to the /bin directory for the tool. The database query_state -option *optionName* command can be used to list the value of the named option when the database state was created.

**-state** *databaseState*
Exports option settings that were in place when the specified database state was run rather than the current option settings.

## export report

Exports reports to a directory.

**export report -list** [**-verbose**]

**export report** [**-path** *directoryPath*] *reportName*|**-all**

**-list** [**-verbose**]
: Lists the available report files for export; if -verbose is included, prints a short description of each report file following its name.

**-path** *directoryPath*
: Specifies the full path to the directory where the report or reports are exported; default is the current directory.

*reportName*|**-all**
: The *reportName* argument specifies an individual report to be exported from the current database (use export report -list to show the available reports); the -all argument exports all of the available reports from the current database state.

## export runtime

Exports instrumentation data to a directory for debugging.

If you transfer exported files to another location, you must copy all exported files. This includes encrypted files, as information in these files might be required for successful bringup and debug. For example, the encrypted cfg file is required for bringup, in addition to the bit file.

**export runtime**

[-debug_encrypt_sources *password*]
[-debug_export_sources **1|0**]
[**-path** *directoryPath*]

-debug_encrypt_sources *password*
: Encrypts the original sources using the given password. During runtime, the user has to enter the password to view the RTL view.

-debug_export_sources **1|0**
: Enables or disables a copy of the original source files into debug project paths. The default is 0.

**-path** *directoryPath*
: Specifies the target directory to export the files required for debugging.

## export srs_bypass

Generates Tcl scripts to update affected FPGAs and to re-run synthesis during an incremental SRS bypass compile flow. Based on the modifications done to the modules, an NLE file is created. This file contains the replace commands for the views in individual FPGAs within the FPGA database.

-**compile_state** *cloned_compile_node_name*

The cloned compile state.

-**path** path

Path to the exported log location.

-**use_mapper**

Find the parent view when the child is in a non-incremental view. The tool traverses through the hierarchy and finds a view that can be replaced. This option also gives an idea whether the linking can be a success or not, before the actual linking happens to individual FPGAs.

## export_verdi_de

Exports the correlation data base (CRDB) and data expansion (DE) setup.

To ensure compatibility, you must use the same VCS_HOME that was used to generate the UC2 database when using the -rtldbdir and same VERDI_HOME that was used to generate the CRDB. Change in versions can result in errors during data expansion.

**export_verdi_de**
**-createlinks**
**-from** <*dir*>
**-rtldbdir**
**-to** <*dir*>

**-createlinks**
Create links to the file.

**-from** *dir*
Directory with the CRDB and DE setup.

**-rtldbdir**
Absolute path of the VCS RTL simv.daidir to be copied. To ensure compatibility, use the same VCS_HOME that was used to generate the UC2 database in the HAPS-Verdi flow.

**-to** *dir*
Destination directory for the exported CRDB and DE information. Use multiple -to arguments to specify multiple export destinations.

### Examples

1. Export CRDB and DE setup from myverdi directory to myde directory.

```
export_verdi_de -from myverdi -to myde
```

2. Export CRDB and DE setup from myverdi directory to myde directory and create links to files.

```
export_verdi_de -from myverdi -to myde -createlinks
```

3. Export CRDB and DE setup from directory myverdi to directories myde1, myde2.

```
export_verdi_de -from myverdi -to myde1 -to myde2
```

## export verification

Exports the complete setup required to run formal verification for a design from the current database state. You can generate the setup from the compile, pre-map, map, and system generate states when the verification_mode option is set to 1.

Once exported, do not move the verification data, or Formality might not work correctly. It is best to run verification with the launch formality command (*launch formality*, on page 70) from within the prototyping tool. For information about running Formality, see Chapter 8, *Verifying the Design* in the *User Guide*.

**export verification [-check_points]** [**-path** *directoryPath*]

**-check_points**
Exports data for check points.

**-path** *directoryPath*
Specifies the target destination directory for the exported verification data from the current database state. The default path is the current directory.

## export vivado

Exports the files needed to run Xilinx Vivado place-and-route on the current database state (see also launch vivado page 70) or returns the path to the Vivado Tcl script.

If you transfer exported files to another location, you must copy all exported files. This includes encrypted files like the .cfg file, which contains LUT configuration information. This file is required for HAPS system bring-up and configuration, in addition to the bit file. The .cfg file is generated for each FPGA for each run.

**export vivado**

[**-format** *netlistType*]
**-path** *directoryPath*
[**-vivado_option_file** *TclFilename*]
[**-vivado_template** *TclTemplate*]

**-format** *netlistType*
Limits the netlist files exported to *netlistType* (edif or verilog). Default id edif.

**-path** *targetDirectory*
Specifies the target directory for the exported Vivado files.

**-vivado_option_file** *TclFilename*
Specifies an optional Tcl file to source from within generated run_vivado Tcl script.

**-vivado_template** *TclTemplate*
Specifies a Vivado Tcl template used to generate Vivado script. Default is *$LIB/xilinx/vivado/run_vivado_haps.tcl*.

# formal_verify

Runs formal verification with Formality and VC Formal on the specified files.

**formal_verify -path** *dir*
**[-bg]**
**[-create_set_reset]**
**[-distribute_cmd** *value***]**
**[-fsc** *value***]**
**[-list_jobs** *value***]**
**[-max_parallel_jobs** *value***]**
**[-post_premap]**
**[-post_compile]**
**[-product** *value***]**
**[-run_jobs***value***]**
**[-semantics** *value***]**

**-path** *value*
Specifies the directory for the exported formal verification data.

**-bg**
Runs in background; immediately returns prompt.

**-create_set_reset**
Automatically creates set and reset for the VC Formal software.

**-distribute_cmd** *value*
Prefix command used for remote job distribution. Accepts any prefix name but it must be valid in shell environment. E.g. cdpl, qsub, lsf.

**-fsc {formality: "fm_setup.tcl", vcformal:"vcf_setup.tcl"}**
Specifies the formal setup constraints (FSC) file that contains constraints for the formal verification run.

**-list_jobs** *value*
Lists the available formal verification jobs. Supports (*?) wildcards in searches.

**-max_parallel_jobs** *{formality:value,vcformal:value}*
Sets the maximum limit for parallel formal verification jobs for Formality and VC Formal.

**-post_premap**
Runs pre-map verification.

**-post_compile**
Runs post-compile verification.

**-product formatlity | vcformal**
Specifies the formal verification tool to use: Formality or VC Formal. The default uses both as required.

**-run_jobs** *value*
Runs the formal verification jobs. Supports (*?) wildcards in searches.

**-semantics simulation | synthesis | auto**
Specifies the formal verification methodology. It can be simulation or synthesis or can automatically run both. This is currently only supported for VC Formal.

## Examples

1. Launch formal verification products on the exported verification data:

```
formal_verify -path verif_check_points -post_premap
-max_parallel_jobs {formality:4,vcformal:3} -fsc
{formality:fm_setup.tcl,vcformal:vcf_setup.tcl}
```

2. Launch formal verification products on the exported verification data using job search:

```
formal_verify -path verif_check_points -post_premap
-max_parallel_jobs {formality:4,vcformal:3} -fsc
{formality:fm_setup.tcl,vcformal:vcf_setup.tcl} -run_jobs {*sggcc*
*group_1*}
```

3. Launch formal products on the exported verification data, for compile output:

```
formal_verify -path verif_check_points -post_compile
-max_parallel_jobs {formality:4,vcformal:3,vcs:10}
```

4. List the GCC job to be run with formal verification on the exported verification data:

```
formal_verify -path verif_check_points -post_premap -list_jobs
*sggcc*
```

5. List all jobs to be run with VC Formal on the exported verification data:

```
formal_verify -path verif_check_points -post_premap -list_jobs *
-product vcformal
```

# generate import_vivado_ip

Integrates Vivado IP into a design according to the type of IP being imported. Types are grouped into two basic categories: Interface IP which uses the *white box* mode and Data-Path IP which uses the *absorb* mode. For detailed information, see *Importing Vivado IP*, on page 344 in the *ProtoCompiler Compiler and Mapper Guide*.

## Syntax

**generate import_vivado_ip**

[**-dcp** *filename*.**dcp**]
[**-dir** *ipDirectory*]
[**-edif**]
[**-list** *ipListFile*]
[**-mode absorb**|**white_box**]
[**-path** *ipDirectory*]
[**-synthesize**]
[**-vm**]
[**-xci** *filename*.**xci**]
[**-xcix** *filename*.**xcix**]

**-dcp** *filename*.**dcp**
Specifies the name of the synthesized checkpoint (DCP) file.

**-dir** *ipSourceDirectory*
Specifies the name of the directory containing multiple IP blocks to be imported. At least one IP block must be included in the directory.

**-edif**
Generates the output in .edif format.

**-list** *ipListFile*
Specifies a file containing a list of IP blocks to be imported. Each block must be on a separate line.

**-mode absorb**|**white_box**
Selects the import mode. The white_box argument allows the IP netlist to be used for timing estimations. In this mode, the IP is not optimized and appears as an empty box in the synthesized netlist. A DCP file, which contains all of the original design constraints, is used to optimize the design around the IP.

The absorb argument allows the IP netlist to be optimized. In this mode, the IP is absorbed into the final synthesized netlist, and the original XDC file is added to the place-and-route options file.

**-path** *ipTargetDirectory*
Specifies the target directory location for the generated IP files. The default is the current directory.

**-synthesize**
Always runs Vivado synthesis on the IP prior to import.

**-edif**
Generates the output in .vm format.

**-xci** *filename*.**xci**
Specifies the name of the Xilinx IP core file.

**-xcix** *filename*.**xcix**
Specifies the name of the Xilinx IP core file.

# help

Displays online help for commands available from the shell prompt or available in a partitioner constraints file.

## Syntax

**help** *commandName*|**pcf_syntax** [*commandName*]

*commandName* **-help**

*commandName*
The name of the requested command. Entering the first part of a command string (for example, run) displays syntax help for all of the run commands. Entering help with no argument lists all of the available shell commands.

To get a list of available options, type option help instead of help option.

For help on TSS board commands, first open a TSS shell with launch tss and then type the help command in that window.

**pcf_syntax** [*commandName*]
Lists the syntax for the partitioner constraints file (PCF) commands. Excluding the *commandName* argument lists the syntax of all PCF commands, and including a *commandName* argument lists only the syntax for the specified PCF command.

# history

Returns a numbered list of executed Tcl commands.

## Syntax

**history**

[**clear**]
[**event** *number*]
[**info** *number*]
[**keep** *number*]
[**nextid**
[**redo** *number*]

**clear** - Clears the history list.

**event** *number* - Returns command *number* from the history list.

**info** [*number*] - Returns the last *number* of commands. If no *number* is included, returns all.

**keep** [*number*] - Sets the number of commands to save in history. Also returns the current setting.

**nextid** - Returns the index number that the next command will be assigned to in the history list.

**redo** [*number*] - Executes the *number* command. If no *number* is given, executes the latest command.

## Examples

```
history event 12
```

```
history redo 4
```

# job

Monitors or queries pending, running, or completed jobs.

## Syntax

**job list** [**-all**]

**job query** *jobID* **-job_name**|**-run_state**|**-return_code**|**-run_time**

**job wait** *jobID* [*jobID* **...**]

**job kill** *jobID* [*jobID* **...**]

**list** [**-all**]
Lists jobs by job ID. If no argument is included, only the job IDs of the running jobs are listed. Including the -all argument additionally lists completed jobs.

**query** *jobID* **-job_name**|**-run_state**|**-return_code**|**-run_time**
Queries specific jobs by job ID. The types of queries are mutually exclusive.

- -job_name – returns the job name string for the specified job.
- -run_state – returns the current state of the specified job; states are running, complete, or canceled.
- -return_code – process return code of the specified job. Returns 0 for jobs that are not complete.
- -run_time – returns the elapsed run time, in seconds, for a completed job; returns the accumulated time if the job is running.

**wait** *jobID* [*jobID* **...**]
Waits for the specified job or jobs to complete before continuing Tcl execution. Multiple job IDs can be specified for the wait list; entries must be separated by a space. When multiple job IDs are included, waits until all jobs are complete.

**kill** *jobID* [*jobID* **...**]
Kills the specified job or jobs. Multiple job IDs can be specified for the kill list; entries must be separated by a space. Querying the run state of a killed job returns canceled.

## Description

Job IDs are automatically created for each run command and reported by a job list command. The other job commands use the job ID to identify the individual jobs to monitor or to query their job status.

## Examples

```
job query job654 -run_time
14
```

# launch

Starts accessory software tools outside of the design project. Launch commands do not alter or change the state of the design. This is a list of the individual launch commands:

| | | |
|---|---|---|
| launch fdc | launch formality | launch idc |
| launch protocompiler | launch tss | launch uc |
| launch vcf | launch vcstatic | launch verdi |
| launch vivado | | |

## launch fdc

Launches the constraint editor.

### Syntax

**launch fdc** *filename* [**-mode gui**]

*filename*
The name of an existing constraint file to open for editing.

**-mode gui**
When -mode is gui (the default), opens the constraint file in the graphical constraints editor.

## launch formality

Launches the Formality formal verification comparison engine to prove the equivalence between two designs. The Formality engine is only supported by the ProtoCompiler product.

For information about running formal verification, see Verifying the Design, on page 937 in the *User Guide*.

### Syntax

**launch formality**

**-path** *exportDirectory*
[**-bg**]
[**-max_parallel_jobs** *value*]
[**-mode gui|batch**]

**-path** *exportDirectory*
Exports the data from the current database state to the specified directory location and runs formal verification on the exported data.

**-bg**
Immediately returns prompt. Multiple jobs run concurrently.

**-max_parallel_jobs** *value*
Specifies number of parallel Formality jobs. Consumes one license for each job.

**-mode**
Specifies Formality launch mode.When the argument is gui the Formality graphical interface is opened. When the -mode argument is batch (the default).

## launch idc

Launches the instrumentor in a graphical window or in batch mode.

### Syntax

**launch idc** *filename* [**-mode gui** |**tclbatch**]

*filename*
The name of the IDC or Tcl batch file to open.

**-mode gui** |**tclbatch**
When the -mode argument is gui, opens the instrumentor graphical interface and reads in the specified IDC file. When the -mode argument is tclbatch (the default), runs the instrumentor UI flow from the specified tcl batch script. Currently, only the tclbatch mode is supported.

## launch protocompiler

Launches the ProtoCompiler tool from within the shell to run Tcl scripts that implement partitioned FPGAs. You can run invocations in parallel. Using launch protocompiler (rather than a separate session) can automatically check out additional licenses as required for parallel runs.

## Syntax

**launch protocompiler**

[**-bg**]
[**-ip_license_wait** *nbrSeconds*]
[**-license_wait** *nbrSeconds*]
[**-max_parallel_jobs** *integer*]
[**-run_dir** *directoryPath*]
[**-script** *tclScriptName*]
[**-tclcmd** *number*]

**-bg**

Starts the tool and immediately returns the shell prompt. Multiple jobs are run concurrently.

**-ip_license_wait** *nbrSeconds*

The launched tool waits for the specified number of seconds for DesignWare licenses, if they are not initially available. If no value is given with -ip_licenes_wait, the tool waits indefinitely.

**-license_wait** *nbrSeconds*

The launched tool waits for the specified number of seconds for licenses, if they are not initially available. If no value is given with -license_wait, the tool waits indefinitely.

**-max_parallel_jobs** *integer*

The number of jobs each script passed can run in parallel. The default is 4. Each license allows up to four jobs (a combination of synthesis and compile-point processes) to run in parallel. The value specified here overrides the default and applies to all -script arguments that follow. For example, to specify that two FPGAs, each with four jobs, be processed in parallel (eight jobs total), use these arguments:

```
-max_parallel_jobs 4 -script fpga1_script.tcl -script fpga2_script.tcl
```

**-run_dir** *directoryPath*

Specifies the run directory for script execution (defaults to the location of script being executed). Applies to all -script arguments that follow.

**-script** *tclScriptName*

Lists a Tcl script to run. Repeat the argument to specify multiple scripts to run in parallel. For example, if you have scripts for two FPGA synthesis runs to run in parallel, specify -script fpga1_script.tcl -script fpga2_script.tcl.

**-tclcmd** *number*

Specifies Tcl code to execute before launching the script. This is a positional argument: it applies to scripts that follow. If given multiple times, subsequent arguments override earlier ones. In this way, different tclcmd arguments can be specified for different scripts.

## Examples

Launch multiple scripts from the same directory:

```
launch protocompiler -script {fpga1/1.tcl fpga2/2.tcl fpga3/3.tcl}
```

Launch multiple scripts from their respective directories, running four jobs in parallel for all the scripts.

```
launch protocompiler -parallel 4 -script path1/a.tcl
   -script path2/b.tcl -script path3/c.tcl
```

Launch two scripts, with the first (`1.tcl`) allowed max_parallel_jobs synthesis jobs in parallel and the second (`2.tcl`) is allowed eight synthesis jobs:

```
launch protocompiler -tclcmd {set var1 1; set var3 3} -script
fpga1/1.tcl -max_parallel_jobs 8 -tclcmd {set var 2; set var1 0}
-script fpga2/2.tcl
```

Launch multiple scripts from relative directories, running four jobs in parallel for all the scripts.

```
launch protocompiler -max_parallel_jobs 4 -script path1/a.tcl
-script path2/b.tcl -script path3/c.tcl
```

Launch two scripts, the first (1.tcl) is allowed the default four synthesis jobs and the second (2.tcl) is allowed eight synthesis jobs:

```
launch protocompiler -tclcmd {set var1 1; set var3 3} -script

fpga1/1.tcl -max_parallel_jobs 8 -tclcmd {set var 2; set var1 0}
-script fpga2/2.tcl
```

Wait for license.

```
launch protocompiler -license_wait 180
```

## launch tss

Launches the target system specification (TSS) shell in either interactive mode or in batch mode. The target system specification shell is only supported by the ProtoCompiler product.

For information about using TSS commands, see *Defining the System in a TSS File*, on page 184 in the *User Guide*.

### Syntax

**launch tss** [*tssFilename*] [**-mode shell**|**batch**]

*tssFilename*
The name of the TSS file to source on startup.

**-mode shell**|**batch**
The startup mode (-mode shell opens the shell in interactive mode, and -mode batch opens the shell in batch mode). Default is shell.

## launch uc

Launches the unified compiler as specified in the -utf argument. The results after running UC are put into the directory specified by -ucdb. The UCDB directory can then be compiled by run compile to create a compile state. Write permission for the current working directory where the unified compiler runs is required.

### Syntax

**launch uc -utf** *filename* **-ucdb** *databaseDirectoryName* **[-v 1.0|2.0]**

**-utf** *filename*
The UTF file to control unified compiler. This includes input files and options.

**-ucdb** *databaseDirectoryName*
The output directory for the unified compile database.

**-v**
The version of unified compile flow. Valid values are 1.0 and 2.0 (default).

## launch vcf

Launches the VC Formal software for formal verification.

### Syntax

**launch vcf** [**-bg**] **[-max_parallel_jobs** *Number***]** [**-mode** *type*] [**-path** *path*]

**-bg**

Immediately returns to prompt. Multiple jobs are run concurrently.

**--max_parallel_jobs** *Number*

Specifies number of parallel VC formal jobs. It consumes 1 license for each job.

**-mode** *type*

Specifies the VC Formal launch mode. The options are **gui** or **batch**. The default is **batch.**

**-path** *path*

The directory for the exported results.

## launch vcstatic

Launches the VC Static software for static analysis.

### Syntax

**launch vcstatic** [**-bg**] [**-dbdir** *path*] [**-show_log**]

The tool runs static analysis and generates a generates a cdc.pcf file with constraints for partitioning. The tool also applies the syn_preserve 1, syn_replicate 0, and syn_allow_retiming 0 attributes to all synchronization registers.

**-bg**

Immediately returns prompt. Multiple jobs are run concurrently.

**-dbdir** *path*

Specifies the absolute path to the CPDB database.

**-show_log**

Monitor log file in window. Available in GUI only.

### Example

```
launch vcstatic -dbdir vccdc.daidir/run_vcsgcdc_cpdb
```

## launch verdi

Launches the Verdi software for RTL debug on the exported debug data.

The launch_verdi version of the command (with the underscore) is used to launch the Verdi software from the runtime executable. The arguments varies for these two commands. See the launch_verdi command for details.

### Syntax

**launch verdi | launch_verdi**
**[-bg]**
**[-crdbgen]**
**[-fast_crdbgen]**
**[-fsdbtype edif | verilog]**
[**-gui**]
**[-rtldbdir** *path*]
[**-run_dir** *path*]
**[-show_log]**

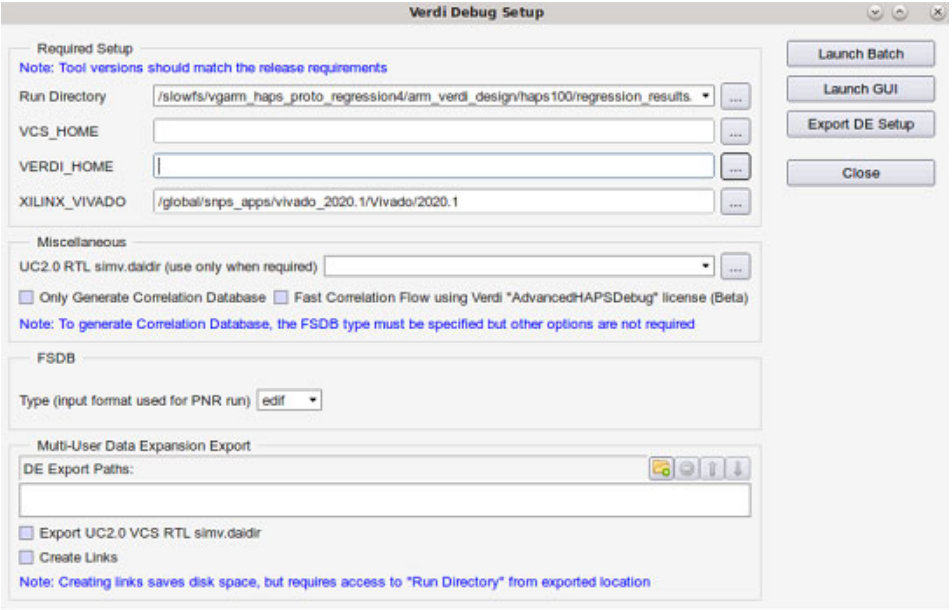
**-bg**
Immediately returns to prompt. Multiple jobs are run concurrently.

**-crdbgen**
Generates only the correlation database.

**-fast_crdbgen**
(*This feature requires a special license. Contact Synopsys support.*) Fast correlation flow using the Verdi tool. Up to 4X faster with CRDB generation.

**-fsdbtype edif | verilog**
Specifies the FSDB type, which can be EDIF or Verilog.

**-gui**
Launches the Verdi GUI. If this argument is not included, the tool runs in batch mode.




**-rtldbdir** *path*

Absolute path to the VCS simv.daidir for the UC flow.
Use the same version of the VCS that was used to generate the UC2 database. Change in versions can cause issues in the flow.

**-run_dir** *path*

The run directory containing the debug data for Verdi.

**-show_log**

Displays the monitor log file in the window. Only valid with GUI mode.

## Examples

1. Launch Verdi tool on the exported debug data for the *system_generate* state using EDIF place-and-route flow:

```
launch verdi -run_dir debug_data -fsdb top.fsdb -fsdbtop top
-fsdbtype edif -de_force_file de_values.txt -gui
```

2. Launch Verdi tool on the debug data exported for the current database state using Verilog place-and-route flow:

```
launch verdi -run_dir debug_data -fsdb top.fsdb -fsdbtop top
-fsdbtype verilog -gui
```

## launch vivado

Launches the Xilinx Vivado place-and-route tool using the specified script.

### Syntax

**launch vivado**

[**-bg**]
[**-run_dir** *extractionDirectory*]
**-script** *tclScriptName*

**-bg**
Runs the specified Tcl script and immediately returns the shell prompt to allow additional jobs to be launched in parallel.

**-run_dir** *extractionDirectory*
The run directory to be used to execute Vivado; the default directory is the script location.

**-script** *tclScriptName*
The name of the script to launch Vivado. Multiple scripts can be entered to run jobs in parallel.

### See Also:

database command, apply_state -import_vivado argument

# message_override

Overrides the severity of a warning or note. For additional information, see *Manipulating Message Display and Reporting*, on page 551 in the *User Guide*.

## Syntax

**message_override**

**-clear**
**-count** *value*
**-error** *ID*
**-limit** [*value*]
**-note** *ID*
**-remove** *value*
**-suppress** *ID*
**-warning** *ID*

**-clear**
Remove all overrides and resets messages to original types.

**-count** *value*
Limits the total number of IDs specified with **-limit** argument. *Value* can be any number **0** or higher, or **unlimited**.

**-error** *ID*
Lists message IDs to treat as errors.

**-limit** [*value*]
Limits the number of messages in the log file. The default is 100. Use with **-count**.

**-note** *ID*
Lists message IDs to treat as notes.

**-remove** *value*
Removes override and resets message type to original value.

**-suppress** *ID*
Lists message IDs to suppress in log file.

**-warning** *ID*
Lists message IDs to treat as warnings.

The command can:

- promote a warning to an error

- promote a note to a warning (or an error)
- demote a warning to a note
- suppress a warning or note from the log file

An actual error message cannot be suppressed or demoted. For notes or warnings, specify log file message IDs to suppress or override as another type (for example, message_override -error MF446 treats the warning message as an error.

## Examples

It is recommended that you set the default limit to something other than unlimited, if possible. To do this, you can specify the following:

```
message_override -limit default -count 1000
```

For unlimited:

```
message_override -limit default -count unlimited
```

More examples are shown below:

1. Upgrade messages with ID MF446 to be treated as an error.

   ```
   message_override -error MF446
   ```

2. Suppress messages with ID BN101 (cannot be done for errors):

   ```
   message_override -suppress BN101
   ```

3. Limit the number of occurrences of messages with IDs MF580 and MF581 to 1000 each in each log file:

   ```
   message_override -limit {MF580 MF581} -count 1000
   ```

4. Unlimit the number of occurrences of messages with ID CL118 in logs and reports:

   ```
   message_override -limit CL118 -count unlimited
   ```

5. Clear existing message overrides:

   ```
   message_override -clear
   ```

6. Change the default limits for all messages:

   ```
   message_override -limit default -count 1000
   ```

# option

A set of commands that specifies and reports value settings for options that define parameters and control flow within a design project. See *List of Options*, on page 82 for a list of the available options.

When multiple options are set, the options are executed sequentially and the last command has precedence.

## Syntax

**option**
**get** *optionName*
**get_default** *optionName*
**help** [*optionName*]
**list**
**reset** *optionName*|**-all**
**set** *optionName optionValue*

**get** *optionName*
Returns the current value for the specified option, not the value of the option when you ran the state. Use database query_state -option *optionName* to get the value of the option at the time that the database state was created. See *List of Options*, on page 82 for the options and their descriptions.

**get_default** *optionName*
Returns the pre-defined default value for the named option. The default value reported is whatever is set after reading all the .synopsys_pc.setup startup files. For more information about using options files, see *Working with Tcl Options Files*, on page 171 in the *User Guide*.

If no value is returned, it indicates that the value is the default value.

**help** [*optionName*]
Displays one-line help on all options; including the *optionName* argument lists help only for named option. Both formats include the accepted option values and affected database states. No legal values will be shown if the set of valid values are large, infinite (integers, filenames, or percentages), or are dependent on other dynamic settings, for example, valid_speedgrades depends on current technology.

**list**
Returns a space-separated list of the supported option names.

**reset** *optionName*|**-all**

Resets the named option to its pre-defined default value; the -all argument resets all options to their default values.

**set** *optionName*

Specifies a value to be set for the defined option. See *List of Options*, on page 82 for the options and their descriptions. Set values apply to processes as they are invoked. Either preset options in setup files or enter them directly at the tool prompt.

## List of Options

The following table lists the available options for the *optionName* argument. Click on an option name in the table to view the corresponding description. Options can also be set from a graphical interface available from any database state; click the Options Editor icon in the Data View (see Options Editor in the *Reference Manual*).

| | | |
|---|---|---|
| advanced_uram_features_on | allow_duplicate_modules | auto_constrain_io |
| auto_export_reports | auto_infer_blackbox | automatic_compile_point |
| beta_vfeatures | beta_vhfeatures | bindandforce |
| cdpl | clock_scaling | compile_strategy |
| compiler_compatible | compress_netlist_for_vivado | condparams |
| continue_on_error | dc_root | design_flow |
| disable_io_insertion | distributed_compile | distributed_global_opt |
| distributed_synthesis | dw_foundation | dw_minpower |
| dw_stop_on_nolic | enable_egcc | enable_global_opt_autodissolve |
| enable_insert_oddr | enable_parallel_area_est | enable_prepacking |
| enable_synenc_partition | enable_upf_map_retention_cell | fix_gated_and_generated_clocks |
| flow_runtime_opt_constraints | force_async_genclk_conv | haps_clockgen_mode |
| hier_verification | hier_verification_percent | incremental_debug |

| | | |
|---|---|---|
| latchram | libext | looplimit |
| max_parallel_jobs | max_parallel_par_explorer | maxfan |
| merge_inferred_clocks | multi_file_compilation_unit | no_constraint_check_in_premap |
| no_sequential_opt | no_sequential_opt_bram_mapping | optimize_ngc |
| pipe | pnr_preserve_regs | prepare_readback |
| ram_insert_oob_check | remove_clock_from_data | resolve_multiple_driver |
| resource_sharing | retiming | rw_check_on_ram |
| sm_higheffort | speed_grade | support_implicit_init_netlist |
| support_xpm | supporttypedflt | synthesis_onoff_pragma |
| synthesis_strategy | technology | valid_technologies |
| valid_speed_grades | verdi_mode | verification_mode |
| verify_check_points | vhdl2008 | view_report_on_error |
| vlog_std | write_verilog | |

**advanced_uram_features_on 1|0**

Enables advanced URAM packing. Set the license-environment variable setenv FpgaInternalBeta 1. By default, this option is enabled (option set advanced_uram_features_on 1).

**allow_duplicate_modules 1|0**

(Compile state)
Allows the use of duplicate module names in Verilog designs. When enabled (option set allow_duplicate_modules 1), the last definition of the module is used and any previous definitions are ignored. The equivalent GUI name for this option is Allow Duplicate Modules in HDL.

**auto_constrain_io 1|0**

(Map and System Generate states)
Determines if default constraints are used for I/O ports that do not have user-defined constraints. When disabled (option set auto_constrain_io 0), only define_input_delay or define_output_delay constraints are considered during synthesis or forward-annotated after synthesis. When enabled,

the software considers any explicit define_input_delay or define_output_delay constraints, as before. The equivalent GUI name for this option is Auto Constrain IO.

**auto_export_reports 1|0**

(All states)
Determines if text versions of log files and reports are exported from the current database state. When enabled (option set auto_export_reports 1), text versions of the available reports and log files are exported to d*atabaseName*_reports adjacent to the database. The equivalent GUI name for this option is Auto Export Reports.

**auto_infer_blackbox 0|1|2**

(Compile and Pre-instrument states)
Determines action taken when an undefined Verilog module is encountered. The equivalent GUI name for this option is Auto-Infer Blackbox.

- When 0 (the default), an undefined module causes the compiler to error out.
- When 1, a black box is created for the undefined module and a warning is issued.
- When 2, a black box is created with bi-dir ports for the undefined module and a warning is issued.

**automatic_compile_point 1|0**

(Map state)
When enabled (option set automatic_compile_point 1), uses the automatic compile point flow to create RTL partitions that can be independently analyzed and mapped in parallel using multiprocessing. The equivalent GUI name for this option is Auto Compile Point.

**beta_vfeatures 1|0**

(Compile state)
Enables Verilog beta features in the compiler (option set beta_vfeatures 1). The equivalent GUI name for this option is Beta Features for Verilog.

**beta_vhfeatures 1|0**

(Compile state)
Enables VHDL beta features in the compiler (option set beta_vhfeatures 1). The equivalent GUI name for this option is Beta Features for VHDL.

**bindandforce 1|0**

(Compile state)
Enables bind and force functions in Verilog (option set bindandforce 1) to either bind a SystemVerilog assertion file without requiring modification

of the original Verilog or VHDL code or to force assignment of a new value to a register or net. The functions are used only for design verification and cannot be synthesized. The equivalent GUI name for this option is Support Bind and Force.

**cdpl 1|0**

(All states, Linux only)
When enabled (option set cdpl 1), runs distributed processing using the list of remote hosts in the common distributed process library. The location of the hosts information file is specified with the CDPL_FPGA_HOST environment variable. For detailed information, see Using CDPL for Distributed Processing, on page 500 in the *ProtoCompiler User Guide*.
The equivalent GUI name for this option is Distributed Processing on Remote Machines.

**clock_scaling off|single_root|multi_root|same_ratio**

(All states)
When enabled, identifies the fastest clock in the HAPS-80 system and generates a report of the scaled clock frequencies during the System-Level Timing Analysis (SLTA). It requires the FpgaInternalBeta license. All the synchronous external clocks are scaled using the same ratio. This ratio is defined by the clock, with respect to the worst ratio in the design. Each asynchronous clock is scaled independently. The default is off. see Scaling External Clock Speed, on page 47 in the *ProtoCompiler User Guide*. The modes of clock scaling are:
single_root - This mode is used with a design in which every clock tree is driven by single, external, synchronous clock.
In case a design consists of external clocks that are synchronous to each other, use set_clock_groups -asynchronous to declare them to be asynchronous.
multi_root - This mode is used with a design in which multiple external clocks are synchronous with each other. All the external clocks that are synchronous with each are scaled using the same ratio and each synchronous clock is scaled independently.
same_ratio: This will also handle designs will multiple external clocks that are synchronous to each other. All the external clocks that are synchronous with each are scaled using the same ratio.

**compile_strategy base|fast|ttfp**

(Compile state)
Sets the strategy for compile stage.
When base is set, this option performs additional optimizations during compile stage to provide better QoR, but with longer runtimes than other modes. When fast is set, this option reduces optimizations to

provide faster compile time at a potential cost of reduced QoR. When ttfp is set, this option further reduces the optimizations to provide fastest compile time at a potential cost of reduced QoR. The default is ttfp.

**compiler_compatible 1|0**

(Compile state)
When enabled (option set compiler_compatible 1) prevents the pushing of tri-states across process/block boundaries. The equivalent GUI name for this option is Conservative Tristate Push Through.

**compress_netlist_for_vivado 1|0**

(Export Vivado)
Controls output file compression. When enabled (option set compress_netlist_for_vivado 1), netlist files that Vivado can read directly in compressed format, for example the main EDIF file, are compressed by export vivado. Use this option to reduce disk usage.

**condparams**

(Compile)
Conditional analysis identifiers file.

**continue_on_error 1|0**

(Compile, Pre-Partition, and Map states)
When enabled (option set continue_on_error 1), allows the compiler to continue, if possible, after encountering a non-syntax error within a design unit and to resume compilation with the next design unit without stopping. The equivalent GUI name for this option is Continue On Error.

**dc_root** *path*

(Compile state)
Specifies path to Design Compiler® installation for access to Synopsys foundation and minPower DesignWare® libraries. The default is $SYNOPSYS. The equivalent GUI name for this option is Design Compiler Installation Location.

**design_flow synthesis|partition**

(Root, Compile, Pre-instrument, and Par Explorer states)
Determines the design flow: single-FPGA or multi-FPGA. The equivalent GUI name for this option is ProtoCompiler Design Flow.

**disable_io_insertion 1|0**

(Map state)
Controls I/O insertion. When enabled (option set disable_io_insertion 1), buffers are not inserted on the I/Os. The equivalent GUI name for this option is Disable I/O Insertion.

**distributed_compile 1|0**

(Compile and Pre-instrument states)
When enabled (option set distributed_compile 1), splits a design into smaller sub-designs by creating multiple netlists (SRS files) that can be compiled in parallel. Processing requires one license for each process and can be run on separate machines using the cdpl option. The distributed compilation feature is in Beta status and requires a separate license from Synopsys. The equivalent GUI name for this option is Distributed Compilation (Beta). For more information on using this feature, see *Running Distributed Compile*, on page 503 in the *ProtoCompiler User Guide*.

**distributed_global_opt 1|0**

(Compile state)
Controls distributed global optimization. The default is **0** (disabled).

**distributed_synthesis 1|0**

(Pre-map, Map, and Pre-partition states)
When enabled (option set distributed_synthesis 1), allows individual processes such as applying constraints on demand, applying gated-clock conversions, and mapping to be performed in parallel for single-FPGA designs using the ProtoCompiler tool (this option is not available with the ProtoCompiler DX software). Processing requires one license for per four processes and can be run on separate machines using the cdpl option. The equivalent GUI name for this option is Distributed Synthesis. For more information on using this feature, see Running Distributed Synthesis, on page 506 in the *ProtoCompiler User Guide*.

**dw_foundation 1|0**

(Compile state)
Specifies the source of the DesignWare models. When enabled (option set dw_foundation 1), uses standard DesignWare building blocks from the Synopsys foundation library. If dw_minpower is also 1, minimum power DesignWare building blocks from the Synopsys minPower library overlay the corresponding models from the DesignWare foundation library. The equivalent GUI name for this option is DesignWare Foundation Library.

**dw_minpower 1|0**

(Compile and Map states)
Specifies the source of the DesignWare building blocks. When enabled (option set dw_minpower 1), overlays minimum power building blocks from the Synopsys minPower library over the corresponding DesignWare foundation library building blocks. The equivalent GUI name for this option is DesignWare MinPower Library.

**dw_stop_on_nolic 1|0**
(Pre-partition, Pre-map, and Map states)
Specifies action taken when a Synopsys DesignWare building block is encountered and no license is found. When disabled (option set dw_stop_on_nolic 0, the default), black boxes DesignWare building blocks and continues; when enabled, stops with an error message. The equivalent GUI name for this option is Error out if no DW license found.

**enable_clock_tree_extraction 0|1**
(All states)
Enables clock tree extraction. The default is 1.

**enable_conn_table 0|1**
(Compile state)
Creates a FPGA pin connectivity table to reduce pin congestion. The default is 0.

**enable_distributed_ilm 0|1**
(All states)
Generates FPGA ILM in distributed fashion in estimate timing or time budget flow. The default is 1.

**enable_egcc 0**
Disables eGCC. By default, eGCC is enabled when you set **haps_clockgen_mode 1**.

**enable_global_opt_autodissolve 0|1**
(Compile state)
This option enables the global optimizer to perform automatic dissolving of tiny lower level hierarchies in a large design. Old fdc or pcf files may need to be updated if constraints were specified on dissolved hierarchical objects. The default is 0.

**enable_parallel_area_est 0|1**
(Pre-partition state)
When enabled, pre-partition mapper and area estimation are run in parallel, which can result in better turnaround time in the pre-partition flow. The default is 1.

**enable_prepacking 1|0**
(Map state)
When enabled (option set enable_prepacking 1, the default), prepares the synthesis netlist for advanced LUT combining. The equivalent GUI name for this option is Enable Advanced LUT Combining.

**enable_synenc_partition 1|0**

(Compile state)
When enabled (option set enable_synenc_partition 0, the default), enables partitioning of synenc-encrypted IPs.

**enable_upf_map_retention_cell 0|1**

(Compile, Pre-map, and Pre-partition states)
Ensures that the tool processes the map_retention_cell command properly. You can specify:

- 0 – Parses and ignores the map_retention_cell command. This is the default.
- 1 – Processes the map_retention_cell command. The tool looks for user-defined retention models and issues an error when they cannot be found.

For more information see *Modeling Retention Logic*, on page 847 in the *User Guide*.

**fix_gated_and_generated_clocks 1|0**

(Map and System Generate states)
Performs gated and generated clock optimization when enabled (option set fix_gated_and_generated_clocks 1). The equivalent GUI name for this option is Clock Conversion.

**flow_runtime_opt_constraints 0|1**

Creates relaxed timing constraints to be used during single FPGA synthesis, resulting in a reduction in place and route and synthesis runtime. The frequency that the design can run on the board can be determined by running System Level Timing Analysis (SLTA) with backannotation. This option must be set before running system generate. The default is 0 (disabled). The equivalent GUI name for this option is Runtime Optimized Constraints. See the Setting Options section in the ProtoCompiler User Guide for details on setting the options using the GUI.

**force_async_genclk_conv 0|1**

(Map state)
When enabled (option set force_async_genclk_conv 1), the tool converts gated clocks even if there are asynchronous set/reset signals in generated-clock logic or datapath latches. When disabled (the default), the tool does not convert gated clocks if asynchronous set/reset signals on generated-clock logic are different from the asynchronous signals on the driven datapath latches. The equivalent GUI name for this option is

Force Generated Clock Conversion. See *Defining Clocks for Gated Clock Conversion*, on page 872 in the *ProtoCompiler User Guide* for information about its use and an example.

**haps_clockgen_mode 0|1**
Enables or Disables HAPS Clockgen. By default HAPS Clockgen is disabled. To enable, use the value 1.

**hier_verification** {**off**|**auto**|**manual**|**all**}
(Compile, Pre-map, Map, Pre-partition, and System Generate states) Provides guidance for creating formal verification blocks from within a design. Verification blocks are individual design modules or a group of modules to be verified through Formality. The flow provides a hierarchical approach for performing verification which can result in faster runtimes and improved debugging capabilities. The individual arguments are:

- auto (default) – creates verification blocks based on internal heuristics.
- off – prevents creation of any verification blocks.
- manual – allows a user-defined area percentage to be used to guide verification block creation.
- all – converts all user modules to verification blocks based on block creation criteria.

The equivalent GUI name for this option is Partition for Formality.

**hier_verification_percent** *integer*
(Compile, Pre-map, Map, Pre-partition, and System Generate states) Available only when the hier_verification option is set to manual. This setting controls creation of verification blocks according to a percentage of the total design area and is normally used to divide complex verification blocks into smaller blocks that can be formally verified. Legal values for *integer* range from 2 to 100 percent. The equivalent GUI name for this option is Formality Partition Weight.

**incremental_debug 1|0**
(Compile state only for RTL instrumentation)
Set incremental_debug to 1 to enable the compiler to run incrementally when probe signals are added or removed, which reduces runtime instead of running the design completely. For more information, see Incremental Compile in the User Guide.

Note that you must apply the database state with the command **database apply_state -idc|-idclist** after setting **incremental_debug.** See *database apply_state -idc|-idclist*, on page 31.

**latchram 1|0**

(Compile, Pre-map, Map, Pre-partition, and System Generate states) infer latch RAMs. To determine if your design has latch memories, set latchram to 1. A message (CL302) is generated for each latch RAM that is inferred. If no latch RAMs are inferred a technology implementation is created, otherwise turn off latch RAM support to create a technology implementation.

**libext {.***libextName1 .libextName2* **...}**

(Compile state)
Adds library extensions to Verilog library files included in your design and searches the directory paths specified that contain these files. The list of extensions is enclosed in curly brackets and separated with spaces; the dot prefix is required. The equivalent GUI name for this option is Verilog Library Extensions.

**looplimit** *loopLimitValue*

(Compile state)
Overrides the default compiler loop limit value of 2000 in the RTL and sets a new global default. You can apply limits on a per-loop basis with the Verilog loop_limit or the VHDL syn_looplimit directives for individual loops. The equivalent GUI name for this option is Loop Limit.

**max_parallel_jobs** *integer*

(All states)
Runs multi-processing with compile points to allow the synthesis software to run multiple, independent, compile-point jobs simultaneously. Using this option provides additional runtime improvements for the compile-point synthesis flow. The equivalent GUI name for this option is Maximum Number of Parallel Jobs. For more information, see *Setting Number of Parallel Jobs*, on page 612 in the *User Guide* and *Running Distributed Synthesis*, on page 506 in the *User Guide*.

**max_parallel_par_explorer** *parallelCount*

(Par Explorer state)
Sets the parallel job count for the run par_explorer command (default is 4). Option effective during map database state. The equivalent GUI name for this option is Maximum Parallel Jobs for Exploratory Place and Route.

**maxfan** *integer*

(Map state)
Sets the fanout limit guideline for the current project. Option effective during map database state. The equivalent GUI name for this option is Fanout Guide.

**merge_inferred_clocks 1|0**

(Pre-map, Map, and Pre-partition states)
Merges all inferred clocks clocking inter-FPGA path objects.

**multi_file_compilation_unit 1|0**

(Compile state)
When enabled (option set multi_file_compilation_unit 1), the Verilog compiler uses the compilation unit for modules defined in multiple files. Enabling this option eliminates the need to repeatedly specify import *package* for each RTL file where it is required, for example. The equivalent GUI name for this option is Multiple File Compilation Unit.

**no_constraint_check_in_premap 1|0**

(All states)
When enabled (option set no_constraint_check_in_premap 1), prevents performing a constraint check during the pre-mapper flow. The equivalent GUI name for this option is Disable Constraint Check in Pre-map.

**no_sequential_opt 1|0**

(Compile, Pre-partition, System Generate, Pre-map, and Map states)
Controls the sequential optimizations for the design (unused registers are still removed from the design). When enabled (option set no_sequential_opt 1), sequential optimizations are not performed and delay and area size can increase. With this option enabled, the FSM Compiler is effectively disabled. The equivalent GUI name for this option is Disable Sequential Optimizations. The default is 0.

**no_sequential_opt_bram_mapping**

This command (option set no_sequential_opt_bram_mapping 1) enables block RAM mapping options with disable sequential optimizations.

**optimize_ngc 1|0**

(Map state)
When enabled (option set optimize_ngc 1), NGC/EDIF optimization is performed during synthesis strategy fast mode. The equivalent GUI name for this option is Optimize NGC/EDIF.

**pipe 1|0**

(Map state)
When enabled (option set pipe 1), creates pipeline stages by moving registers into a multiplier to allow designs to run at a faster frequency. The equivalent GUI name for this option is Pipelining.

**pnr_preserve_regs 1|0**

(Global directive)
When enabled (option set pnr_preserve_regs 1), preserves registers in Vivado. The equivalent GUI name for this option is PNR Preserve Registers. This option is disabled by default.
Use syn_pnr_preserve_regs, on page 736 to apply the option to specific modules or views.

**prepare_readback 1|0**

(Map state, before running export vivado)
When enabled (option set prepare_readback 1), allows Xilinx readback during debug phase. Argument value must be set prior to executing export vivado. The equivalent GUI name for this option is Enable Xilinx Readback Capabilities. See *Using Global State Visibility (GSV)*, on page 753 in the *ProtoCompiler User Guide* for details.

**ram_insert_oob_check 1|0**

Enables insertion of extra logic to handle *out-of-bound* memory access. When enabled (option set ram_insert_oob_check 1), you will read 0 from *out-of-bound* locations of the memory instead of x. Set the license-environment variable setenv FpgaInternalBeta 1 to enable this option. The default is 0(disabled).

**remove_clock_from_data 1|0**

(Pre-map state)
When enabled (option set remove_clock_from_data 1), eliminates the latch Data/Set/Reset dependency on the Latch Enable. The default is 1(enabled). For more information, see Conversion of Datapaths with Latches, on page 905.

**resolve_multiple_driver 1|0**

(Compile, Pre-map, Pre-partition, and System Generate states)
When enabled (option set resolve_multiple_driver 1), connects a net driven by VCC or GND and an active driver to the VCC or GND driver. The equivalent GUI name for this option is Resolve Mixed Drivers.

**resource_sharing 1|0**

(Compile state)
Globally controls resource sharing. When enabled (option set resource_sharing 1), arithmetic operators are shared for mutually exclusive statements. This option is a compiler-specific optimization, and does not affect resource sharing in the mapper. To enable or disable individual modules, use the syn_sharing directive. The equivalent GUI name for this option is Resource Sharing.

**retiming 1|0**

(Map state)
When enabled (option set retiming 1), registers can be moved into combinational logic to improve performance. The default value is 0 (disabled). The equivalent GUI name for this option is Retiming.

**rw_check_on_ram 1|0**

(Map state)
When enabled (option set rw_check_on_ram 1), inserts bypass logic around RAMs to prevent simulation mismatches when a read or write conflict exists for the RAM. The equivalent GUI name for this option is Read Write Check on RAM.

**sm_higheffort 0|1**

When enabled (option set sm_higheffort), FSM extraction will be improved.**speed_grade**
Returns or sets the speed grade for individual FPGAs. For partitioned designs, set the speed grade at the system level with the board_system_create command (*board_system_create*, on page 249) in the TSS file.

**support_implicit_init_netlist 1|0**

(Map state)
When enabled (option set support_implicit_init_netlist 1), allows implicit initial-value support for instantiated primitives. The equivalent GUI name for this option is Implicit Initial Value Support for Instantiated Primitives.

**support_xpm 0|1**

When enabled (option set support_xpm1), allows all states.

**supporttypedflt 1|0**

(Compile state)
When enabled (option set supporttypedflt 1), the compiler passes initial values through a syn_init property to the mapper. The equivalent GUI name for this option is Implicit Initial Value Support.

**synthesis_onoff_pragma 1|0**

(Compile state)
When enabled (option set synthesis_onoff_pragma 1), ignores any code enclosed between synthesis on and synthesis off directives and treats these third-party directives as translate_on/off directives (see translate_off/translate_on, on page 869 of the *Compiler and Mapper Guide* for details). The equivalent GUI name for this option is Synthesis Pragma.

**synthesis_strategy advanced|fast|routability|base**

(Map state)
Sets the synthesis strategy.

- When advanced is set, this option performs additional optimizations during logic synthesis to provide better QoR, but with longer runtimes than other modes.
- When fast is set, this option reduces optimizations to provide faster logic synthesis runtimes at a potential cost of reduced QoR.
- When routability is set, this option performs placement-aware congestion techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime. The default is routability.
- When base is set, the software performs as it would with advanced and fast modes turned off. While base will give the best area estimation and timing results, it does not attempt to eliminate routing congestion.

**technology**

Retrieves the technology in use. For example:

```
% option get technology
HAPS-80
```

**valid_speed_grades**

Lists valid values of speed_grade for the current technology.

**valid_technologies**

Returns a list of valid system technologies that can then be specified with the database create command. For example: HAPS-80, HAPS-70, HAPS-DX7_S4 and HAPS-DX7_S6.

**verdi_mode 1|0**

*Supported only on HAPS-80 and HAPS-100*
Enables or disables the automated integration of Verdi mode. The equivalent GUI name for this option is Verdi Mode.

If you set verdi_mode on HAPS-70, the tool displays the following message in the GUI mode and prints it in the Runtime log:
`Error: Failed to apply ll file to design for <FPGA PhysiscalAddress>. Verdi mode is not supported on this system (HAPS-70); it is supported only on HAPS-80 and HAPS-100.`

**verification_mode 1|0**

(Compile, Pre-map, Map, and Pre-partition states)
Enables/disables the Verification Mode option. When enabled (option set verification_mode 1), various sequential optimizations that cannot be easily verified are disabled; for example, the inference of resettable SRLs. The trade-off when you enable the Verification Mode option is that you may sacrifice performance or area, because the optimizations are not performed. The equivalent GUI name for this option is Verification Mode.

**verify_check_points 1|0**

(Compile, Pre-map, Map, and Pre-partition states)
Checks the verification points in the Verification Mode. The verification flow will be tuned to control certain optimizations, which are not formally verifiable by the available formal verification tools such as Formality and VC Formal. For example, global retiming is not formally verifiable. When enabled (option set verify_check_points 1), the verification flow will generate all the check points data and setup required for running multiple formal tools at various stages during synthesis. The tool performs automated logic equivalence checking with minimal to no manual intervention. Use this option with the Verification Mode option (option set verification_mode 1), before the compile state.

**vhdl2008 1|0**

(Compiler state)
Controls the selection of VHDL 2008 compiler functions. When enabled (option set vhdl2008 1), the VHDL 2008 compiler functions are available. The equivalent GUI name for this option is VHDL-2008.

**view_report_on_error 1|0**

(All states)
Automatically opens log files that contain errors when enabled (option set view_report_on_error 1). The equivalent GUI name for this option is View Reports with Errors.

**vlog std sysv|v2001|v95**

(Compiler and Pre-instrument states)
Sets the default Verilog standard for any files where the standard is not explicitly set; valid settings are sysv, v2001, or v95. The equivalent GUI name for this option is Verilog Standard.

**write_verilog 1|0**

(Map state)
When enabled (option set write_verilog 1), writes Verilog mapped netlists. The equivalent GUI name for this option is Write Mapped Verilog Netlist. To view the netlist see *export netlist*, on page 56.

## Description

Defines an option to be instantiated by a subsequent run command. These options apply to the current design project and are retained. Options commands affect the operation of the individual processes when they are invoked.

The tool follows this process when the option command is specified:

- Passes *optionValue* to all processes by default.
- When *optionValue* is set successfully, the value is returned. Notification is given if *optionValue* could not be successfully applied when the corresponding process is subsequently run.

Options can be set in one of five ways, in order of precedence:

1. In the $INSTALL_DIR/*.setup* file
2. In the $USER_HOME/*.setup* file
3. $CWD/*.setup* file
4. In an options Tcl file sourced from the shell prompt
5. Direct application via the option set command entered at the shell prompt

Option commands are persistent. Once a value is set in the shell, the value remains until explicitly set to a different value.

## Example

```
% option get synthesis_strategy fast
partition

% option help cdpl
cdpl:    Distributed processing on remote machines
         valid settings: <0|1>
```

# process_bd_ip

*Xilinx*

Use a Tcl procedure to automatically process the block design (BD) file and convert it to a DCP file in the synthesis tool using batch mode.

## Syntax

**process_bd_ip -bdfile** *bdFileName* | **-repopath** *repositoryPath* **-xboard** *xboardName*

## Arguments and Options

| Option | Description |
|---|---|
| **bdfile** *bdFileName* | BD input file to be processed |
| **xboard** *xboardName* | Specifies the Xilinx board |
| **repopath** *repositoryPath* | Specifies the path to the IP repository |

When the BD file is created using any Xilinx technology:

**process_bd_ip -bdfile** *bdFileName*

When the BD file is created using the Xilinx board:

**process_bd_ip -bdfile** *bdFileName* --**xboard** *xboardName*

When the BD file consists of customized IPs along with the Xilinx IPs:

**process_bd_ip -bdfile** *bdFileName* **-repopath** *repositoryPath*

Multiple IP repositories can be used by using space between the paths:

**process_bd_ip -bdfile** *bdFileName* **-repopath** {*repositoryPath1 repositoryPath2 repositoryPath3}*
**process_bd_ip -bdfile** zusys.bd **-repopath**
{ /Reference_Design/StarterKit/ip_lib/axis_live_audio_1.0/
/Reference_Design/StarterKit/ip_lib/rgpio/ /Reference_Design/StarterKit
/ip_lib/SC0808BF/ }

# report constraint_check

Runs the constraint checker and reports any errors in a Syntax Constraint Check Report file (syntax_constraint_check.rpt), which is displayed in the Report View.

Run the command from the compile, pre-map, map or pre-partition database state. This is a syntax-only mode which does not require design information, can be run from any database state. Executing the command automatically displays the report when complete; the recommended flow is to continue to generate reports and modify the constraints until all warnings are resolved.

## Syntax

**report constraint_check**
[**-bg**]
[**-fdc** *fdcFilename*|*tclListofFDCfiles*]
[**-fdclist** *fdcListFilename*]
[**-out** *subStateName*]
[**-syntax_only**]

**-bg**
Runs constraint checker in the background. Specify this option to run pre-partition in parallel to constraint checker.

**-fdc** *fdcFilename*|*tclListofFDCfiles*
Specifies the name of an FPGA design constraints file to be compiled with the design. The *fdcFilename* argument is a string value that specifies the path name to the constraint input file; the string value *tclListofFDCfiles* is a standard Tcl list of one or more constraint files.

**-fdclist** *fdcListFilename*
Lists the required FPGA design constraints files in a single file (*fdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding FDC file. A filename extension is not required for *fdcListFilename*, but the name of the file must be unique.

**-out** *subStateName*
Specifies a text string identifier to allow individual reports to coexist within the same database state. The report name is formed by combining the name of the database with *subStateName*.

**-syntax_only**

Runs a syntax check on one or more FDC files without requiring design information; this check can be run from any state.

# report ctc

Generates a clock tree conversion report. Use the arguments to select the information written to the file.

## Syntax

**report ctc**
[**-fdc** *fileName*]
[**-fdclist** *fileList*]
[**-gcc 0|1**]
[**-out** *resultFile*]

**-fdc** *fileName*
Include one or more FPGA design constraints (FDC) file.

**-fdclist** *fileList*
Include a file listing one or more FDC files.

**-gcc 0|1**
Executes GCC process.

**-out** *resultFile*
Sets the output file name. Full name will be <parentstate>.<substate>.

# report debug_essential_signals

Identifies essential signal analysis data for debug, and generates files required for unified compile (UC) debug.

The tool generates the following three files in the export directory defined in the command:

- esa_siglist_<*moduleName*>_uc_report.txt—Text file with essential signals to preserve. Signals are generated based on the design hierarchy.
- esa_siglist_<*moduleName*>_uc_dumpvars.sv—System Verilog file with all essential signals added as dumpvars.
- esa_siglist_<*moduleName*>_uc.idc—Template IDC file with essential signals instrumented. Edit the file to update IICE settings and other configuration. Use this IDC at the compile state for instrumentation or use it with the report debug_visibility command to generate the SRS based IDC that can be used later in the flow.

To run this command:

- Use the same version of Verdi and VCS-MX.
- Set the VERDI_HOME and VCS_HOME environment correctly.
- Use the UC 2.0 database generated using option verdi_mode set to 1.

## Syntax

**report debug_essential_signals -export_path** *path* **-top_module** *moduleName* **-ucdb** *ucdbDir* **-rtldbdir** *rtldbDir*

**-export_path** *path*
Specifies the directory path to export essential signals data.

**-top_module** *moduleName*
Design module name for analysis.

**-ucdb** *ucdbDir*
Specifies path to the UC database.

**-rtldbdir** *rtldbDir*
Specifies the path to the VCS sim.daidir when the VCS sim.daidir directory path is different from the one used to generate UCDB. By default, the tool looks for the simv.daidir that is available parallel to the UCDB.

## Examples

- The following is a syntax usage example to launch UC and report debug essential signals:

  ```
  launch uc -utf <utf_file_name> -ucdb <ucdb_dir_name> -v 2.0
  report debug_essential_signals -export_path <path> -top_module
  <moduleName> -ucdb <ucdb_dir_name>
  ```

- In the following example, the essential signal analysis data is reported to a directory named esa_data, for the design dut with UC 2.0 database in the directory ucdb:

  ```
  report debug_essential_signals -export_path esa_data -top_module
  dut -ucdb ucdb
  ```

# report debug_visibility

Reports available netlist signals for debug, based on the provided IDCs with RTL hierarchal path.

Command is available at compile, pre_partition, map, and system_generate database states. *For the map state, you need a Beta license. Contact Synopsys Support.*

## Syntax

**report debug_visibility -export_path** *path* [**-idc** *fileName* **-idclist** *listName*]

**-export_path** *path*
Directory path to export debug visibility reports.

**-idc** *fileName*
IDC file to evaluate.

**-idclist** *listName*
File listing one or more IDC files to evaluate.

## Description

The command does a DRC check and generates a comprehensive report of the signals for the current state and all previous visibility states. The command generates reports only for signals that are yet to be instrumented. Signals that are already instrumented are ignored and are not correlated.

The command shows the required signals, the stage where they are required, and reports whether the signal is available at the required stage. For example, if the command is run from system generate (sg0), you get these reports:

```
dvrpt_system/
   /uchapsdb_sg0_debug_visibility
      /sg0
         debug_visibility_rtl.log
         rtl_FB1_uA_srs.idc
         rtl.idc
```

The command also generates IDC files. At the compile and pre-partition stages, this IDC is based on the compiled database, with RTL signal paths. When run from the system generate stage, the command generates a separate IDC file for each FPGA with the signals that can be instrumented for that FPGA.

The incremental debug option must be set to 1 (enabled) before running the compile state.

When run from the map state, the command checks the correlated netlist names against the registers available on the LL files exported after the Vivado run. The IDC generated is compatible only with the Debug ECO flow. The signal names provided to add signals to the instrumentation will contain the existing register names in the FPGA implementation. The signals delete commands are not checked and are copied directly from the input IDC. If the command is run from the map state, you get these reports:

```
/visibility_reports
   /srs_m0_debug_visibility
      /m0
         debug_visibility_siglist.log
         siglist_ECO.idc
```

The incremental debug option must be set to 1 (enabled) before running the map state.

Note that the feature does not support partial instrumentation in RTL IDC.

## Examples

To report and export the debug visibility data to the dvrpt directory, for IDC list siglist.idc:

```
report debug_visibility  -export_path dvrpt  -idc siglist.idc
```

## Examples of IDC Files

This is an example of an RTL IDC file used as input to the report debug_visibility command:

```
set iiceName "IICE_UC"
device jtagport umrbus
iice new $iiceName -type regular
iice controller -iice $iiceName none
iice sampler -iice $iiceName -depth 128
iice sampler -iice $iiceName -pipe 3

### Define the IICE clock
set clockName "clock"
iice clock -iice $iiceName -edge positive $clockName

signals add -iice $iiceName -silent -trigger -sample {top.state}
signals add -iice $iiceName -silent -trigger -sample {top.op}
```

This is an example of a post-compile (SRS) IDC generated by the command:

```
############# Debug Visibility Based SRS IDC ####################
#
#  Following signals are generated based on correlating the given
#  RTL signals to those available in FB1.uA
#  These signals needs to be modified as appropriate, based on the
#  required debug methodology
#
################################################################

## Sample IICE setup; needs to be changed as per hardware
requirements
#Please change line below to prevent CAPIM address clash
#device capimbaseaddr 8
# Mode and uc_groups are set to the recommended values.
# Ensure to check help docs and change them accordingly.

iice new {IICE_UC} -type regular -mode {none} -uc_groups {}
iice controller -iice {IICE_UC} none
iice sampler -iice {IICE_UC} -depth 128
iice clock -iice {IICE_UC} -edge positive   {SRS.clock}
signals add -iice {IICE_UC}  -sample -trigger  {SRS.state[31:0]} \
{SRS.op[3:1]}
```

This is an example of input and output IDC files when the command report debug_visibility -export_path dvrpt -idc siglist.idc is run at map state:

Content of the input IDC file *siglist.idc:*

```
set iiceName "IICE_UC"

device jtagport umrbus
```

```
iice new $iiceName -type regular
iice controller -iice $iiceName none
iice sampler -iice $iiceName -depth 128
iice sampler -iice $iiceName -pipe 3

signals add -iice $iiceName  -trigger -sample
{eight_bit_uc.decode.opcode_goto}
signals add -iice $iiceName  -trigger -sample
{eight_bit_uc.decode.opcode_call}
signals add -iice $iiceName  -trigger -sample
{eight_bit_uc.decode.opcode_retlw}
signals add -iice $iiceName  -trigger -sample
{eight_bit_uc.decode.decodes}
signals add -iice $iiceName  -trigger -sample
{eight_bit_uc.prgmcntr.c_stacklevel}
signals add -iice $iiceName  -trigger -sample
{eight_bit_uc.regs.addr}
signals add -iice $iiceName -trigger -sample
{eight_bit_uc.regs.regfile_out}
```

Content of the output file *siglist_ECO.idc*:

```
######### Debug Visibility IDC for debug ECO ###################
#
#  The following signals are generated based on correlating the
given RTL signals to the registers available
#  for incremental instrumentation. This IDC file is only
compatible with the debug ECO flow.
#
################################################################

# Estimated bit count to be instrumented with debug ECO: 16 bits.
# IICE names available in the design { IICE_UC }.

set base [lindex [instrumentation list] 0]

instrumentation new -instr $base
instrumentation load incr_$base


signals add -iice {IICE_UC}  -sample -trigger
{INCR_FILE.decode.decodes[4:0]} \

{INCR_FILE.prgmcntr.c_stacklevel[2:0]} \
{INCR_FILE.regs.addr[4:0]} \
{INCR_FILE.decode.opcode_retlw} \
{INCR_FILE.decode.opcode_call} \
{INCR_FILE.decode.opcode_goto}
```

Content of the output file *debug_visibility_siglist.log*:

```
************** Debug Visibility Report ************************
Estimated bit count to be instrumented with debug ECO: 16 bits.
-----------------------------------------------------------------

Required signal:     eight_bit_uc.decode.decodes
Available for ECO:   dut_inst.decode.decodes[4:0]
Not available for ECO (make sure the register was not renamed or
removed by the PnR tool. Rerun PnR with a DONT_TOUCH directive on
the register):

   dut_inst.decode.decodes[10]
   dut_inst.decode.decodes[11]
   dut_inst.decode.decodes[12]
   dut_inst.decode.decodes[13]
   dut_inst.decode.decodes[5]
   dut_inst.decode.decodes[6]
   dut_inst.decode.decodes[7]
   dut_inst.decode.decodes[8]
   dut_inst.decode.decodes[9]

-----------------------------------------------------------------
Required signal:     eight_bit_uc.decode.opcode_call
Available for ECO:   dut_inst.decode.opcode_call
-----------------------------------------------------------------
Required signal:     eight_bit_uc.decode.opcode_goto
Available for ECO:   dut_inst.decode.opcode_goto
-----------------------------------------------------------------
Required signal:     eight_bit_uc.decode.opcode_retlw
Available for ECO:   dut_inst.decode.opcode_retlw
-----------------------------------------------------------------
Required signal:     eight_bit_uc.prgmcntr.c_stacklevel
Available for ECO:   dut_inst.prgmcntr.c_stacklevel[2:0]
-----------------------------------------------------------------
Required signal:     eight_bit_uc.regs.addr
Available for ECO:   dut_inst.regs.addr[4:0]
-----------------------------------------------------------------
Required signal:     eight_bit_uc.regs.regfile_out
Not available:       Please recompile module "reg_file", by
preserving this signal

*****************************************************************
Debug Visibility Report
*****************************************************************
```

# report external_tool_versions

Displays third-party external tool information, such as installation environment variable settings and supported tool versions.

## Syntax

**report external_tool_versions** [*tool*] [**-list**] [**-run_requirements**]

*tool*
Shows the version data for the specified vendor tool. Use **-list** to see supported tools. By default, all external tools are shown.

**-list**
Lists the names of tools available in this command.

**-run_requirements**
Shows the version data for all tools required to run the current job flow.

# report instrumentation

Generates the instrumentation report <*topModuleName*>_identify_report.log based on the Instrumentor Debugger Constraints (IDC) files applied up to the point in the flow. The report provides instrumentation details for each FPGA. This report helps you reduce the turnaround time by detecting debugging limitations early in the cycle.

This command is supported only on partition and system_route states.

## Syntax

**report instrumentation**

### Example

Report instrumentation log at partition state:

```
Signal distribution for IICE IICE_0:
FPGA    Width  Type       Instrumentation Name
----------------------------------------------
FB1.uA  3      secondary
        1      S          SRS.reset_n
        1      S&T        top.trigger2
        1      S&T        top.trigger1
FB1.uB  2      secondary
        1      S          SRS.trigger3
        1      S&T        top.trigger4
FB1.uC  3      secondary
        3      S          SRS.up_counter[2:0]
FB1.uD  8      secondary
        8      S&T        top.event_counter
==============================================
Current instrumentation information:   _IICE=IICE 0
      FPGA=FB1.uA
      Total instrumentation in bits: Sample Only 1, Trigger Only 0, Sample and trigger 2, Assertions 0
      Instrumentation in bits:               Sample Only 1, Sample and trigger 2, Assertions 0. All sa
      FPGA=FB1.uB
      Total instrumentation in bits: Sample Only 1, Trigger Only 0, Sample and trigger 1, Assertions 0
      Instrumentation in bits:               Sample Only 1, Sample and trigger 1, Assertions 0. All sa
      FPGA=FB1.uC
      Total instrumentation in bits: Sample Only 3, Trigger Only 0, Sample and trigger 0, Assertions 0
      Instrumentation in bits:               Sample Only 3, Sample and trigger 0, Assertions 0. All sa
      FPGA=FB1.uD
      Total instrumentation in bits: Sample Only 0, Trigger Only 0, Sample and trigger 8, Assertions 0
      Instrumentation in bits:               Sample Only 0, Sample and trigger 8, Assertions 0. All sa
      Maximum sample clock frequency: 80.0000 MHz
exit status=0
```

Report instrumentation log at system-route state:

```
Signal distribution for IICE IICE_0:
FPGA    Width  Type       Instrumentation Name
---------------------------------------------
FB1.uA  3      primary
        1      S&T        top.trigger1
        1      S&T        top.trigger2
        1      S          SRS.reset_n
FB1.uB  2      secondary
        1      S&T        top.trigger4
        1      S          SRS.trigger3
FB1.uC  3      secondary
        3      S          SRS.up_counter[2:0]
FB1.uD  8      secondary
        8      S&T        top.event_counter
=============================================
Current instrumentation information:    IICE=IICE 0
       FPGA=FB1.uA
       Total instrumentation in bits: Sample Only 1, Trigger Only 0, Sample and trigger 2, Assertions 0
       Instrumentation in bits:               Sample Only 1, Sample and trigger 2, Assertions 0. All sar
       FPGA=FB1.uB
       Total instrumentation in bits: Sample Only 1, Trigger Only 0, Sample and trigger 1, Assertions 0
       Instrumentation in bits:               Sample Only 1, Sample and trigger 1, Assertions 0. All sar
       FPGA=FB1.uC
       Total instrumentation in bits: Sample Only 3, Trigger Only 0, Sample and trigger 0, Assertions 0
       Instrumentation in bits:               Sample Only 3, Sample and trigger 0, Assertions 0. All sar
       FPGA=FB1.uD
       Total instrumentation in bits: Sample Only 0, Trigger Only 0, Sample and trigger 8, Assertions 0
       Instrumentation in bits:               Sample Only 0, Sample and trigger 8, Assertions 0. All sar
       Maximum sample clock frequency: 80.0000 MHz
exit status=0
```

# report list

Lists the name or names of reports and log files available from the current database state.

## Syntax

**report list** [**-error_logs**][**-verbose**] **-error_logs**

Lists only the log files with errors.

**-verbose**

Prints a short description of each report/log following its name.

## Example

To list the reports and log files available from a pre-instrument state:

```
% report list
bus_demo_identify_db_generator.srr
```

To include the short description with the report name:

```
% report list -verbose
bus_demo_identify_db_generator.srr: Pre-Instrument Log
```

# report messages

Returns error message data from the specified report.

## Syntax

**report messages** *logFile* [**-id** *value*] [**-out** *value*] [**-severity** *value*]

*logFile*
Specifies one or more log files to query for message details.

**-id** *value*
Restricts report to messages matching this id. Use the * or % wildcard to match any string.

**-out** *value*
Name of the output file to be written rather than writing to the Tcl window.

**-severity** *value*
Limits messages to a specific severity. Can specify one or more messages as an error, warning, note, or advice. Multiple severities can be specified. If none is specified, all types are shown.

## Output Format

Running the report messages command produces a list of error messages. Messages are displayed in the following format with each error message beginning on a new line:

**{***logfileName*:*linenum ID* **{***messageText***}}**

## Examples

1. Query BN132 messages from the test.srr log file.

   ```
   report messages test.srr -id BN132
   ```

2. Query error and warning messages from the test.srr log file.

   ```
   report messages test.srr -severity error warning
   ```

3. Query critical warnings (CW) or downgradable errors (DE). You can search for these messages using wildcards. For example:

```
report messages test.srr -id DE*
```

# report out_of_date

Reports detailed information for the state or the sequence of states about files and options that have changed since the previous run of the current state to a log file.

## Syntax

**report out_of_date** [**-mode current|active_tree**] [**-out** *subStateName*]

**-mode current|active_tree**
Report out of date status for the current state only (current), or the current active tree (active_tree), which means all states back to root from the current state. The default is active_tree.

**-out** *subStateName*
Output substate name. The report name is formed by combining the name of the parent state with the substate name. For example *parentStateName*.*subStateName*.

# report rtl_diagnostics

Compiles the design and reports status in the compiler log file. The command is run from root and automatically displays the report when complete.

For information about using this functionality, see *Running Diagnostic Compiler Mode*, on page 85 in the *User Guide*.

## Syntax

**report rtl_diagnostics**
[**-cdc** *cdcFilename*|*tclListofCDCfiles*]
[**-cdclist** *cdcListFilename*]
**hdl_define** *variable*[**=***value*]]
[**-hdl_param** *parameter***=***value*]
[**-i** | **-include_path {***listofPaths***}**|*pathVariable*]
[**-ilist** *filename*]
[**-l** | **-library_path {***listofPaths***}**]
[**-lib** *libName*]
[**-llist** *filename*]
[- [**-lmf** *tclListofLibMapFilenames*]
[**-out** *subStateName*]
[**-src** *srcFilename*|*tclListofSourceFiles*]
[**-srclist** *srcListFilename*]
[**-srcstate** *databaseState*]
[**-top_module** *moduleName*]
[**-type** *fileType*]
[**-vlog_std** *standard*]

**-cdc** *cdcFilename*|*tclListofCDCfiles*]
Specifies the name of a compiler directives constraints file to be compiled with the design. The *cdcFilename* argument is a string value that specifies the path name to the constraint file; the *tclListofCDCfiles* argument is a standard Tcl list of one or more constraint files. If the -cdclist argument is also used, *cdcFilename* is automatically included with the other listed files

**-cdclist** *cdcListFilename*
Lists the required compiler directives constraints files in a single file (*cdcListFilename*). Each entry within this file is on a separate line and

each entry must specify the path name to the corresponding CDC file. A filename extension is not required for *cdcListFilename*, but the name of the file must be unique.

**-hdl_define** *variable*[**=***value*]
Specifies a Tcl list of HDL defines to be applied to the top-level module.

**-hdl_param** *parameter***=***value*
Specifies a Tcl list of parameters to be applied to the top-level module. You cannot specify a file list with this argument. To specify a list of parameters, either create a file and use an include statement to add the file in the RTL code, or enclose a list of parameters in curly braces as shown below:

```
-hdl_param {W=4, K=5, X=8}
```

**-lmf** *tclListofLibMapFilenames*
Specifies a Tcl list of library mapping files. For more information on library mapping files, see Using UUM and Group Mapping, on page 109 in the *User Guide*.

**-i|-include_path {***listofPaths***}|***pathVariable*
Defines the search path used for ‘include commands relative to the active directory. The argument ***listofPaths*** is a semicolon separated list of the include-file directories enclosed in curly brackets; the ***pathVariable*** is a previously-defined variable for the include paths that you can now include on the command line (see *Examples*, on page 119). The software searches for include files first in the source file directory and then in included-path directory order, stopping at the first occurrence of the include file.

**-ilist** *filename*
A file listing one or more include paths.

**-l|-library_path {***listofPaths***}|***pathVariable*
Defines the search path used for libraries relative to the active directory. The argument *listofPaths* is a semicolon separated list of the library directories or files enclosed in curly brackets (supporting a single library file format allows custom library definitions for VCS file compatibility). The *pathVariable* is a previously-defined variable for the library paths that you can now include on the command line.

**-llist**
A file listing one or more library include paths.

**-lib** *libName*
Specifies the library to use for subsequent files (default is work). The *libName* argument is a string value that specifies the full path name to the indicated library.

**-out** *subStateName*
Specifies a text string to allow multiple reports to coexist within the same database state. The report name is formed by combining the name of the database with *subStateName*.

**-src** *srcFilename|tclListofSourceFiles*
Specifies a single HDL (Verilog or VHDL) input file or a Tcl list of source files. The *srcFilename* argument is a string value that specifies the path name to the source input file; the *tclListofSourceFiles* argument is a standard Tcl list of one or more source files. If the -srclist argument is also used, *srcFilename* is automatically included with the other listed files.

**-srclist** *srcListFilename*
Lists the required HDL (Verilog/VHDL) or edif input source files in a single file (*srcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding input file. When creating a list of source files, the top-level file either must be the last entry in the list of files or explicitly specified with a -top_module argument. A filename extension is not required for *srcList-Filename*, but the name of the file must be unique.

**-srcstate** *databaseState*
Specifies the database states to be compiled in a bottom-up compile flow. The argument is repeated to identify each state to be included. To specify a path to a database state, specify the path and then use a pipe (|) between the path and the name of the database state. See *Running Bottom-Up Compile*, on page 90 in the *User Guide* for a step-by-step procedure.

**-top_module** *moduleName*
Specifies the name of the top module for the design. The *moduleName* string defines the base name for the design output files.

**-type** *fileType*
Specifies the file type applied to subsequent files when the file type is not recognized as any of the predefined file types. The *fileType* argument is a string value; the period extension separator is assumed. The default *fileType* for NGC files is xilinx, and the default *fileType* for EDIF files is edif.

**-vlog_std** *standard*

Specifies the Verilog standard applied to subsequent Verilog files (the -vlog_std argument must precede any -src or -srclist argument). Accepted string values for *standard* are v95 (Verilog 95), v2001 (Verilog 2001), and sysv (SystemVerilog). The VHDL standard is set with the VHDL2008 option (see vhdl2008).

## Report Log

The report rtl_diagnostics command is only valid from root and produces a single report file (*designName*_compiler.srr). The report displayed in the Report View is named RTL Diagnostics Report.

## Examples

```
report rtl_diagnostics -src prep2_2.v -cdc myconsfile.cdc

report rtl_diagnostics -src "$RTL_PATH/core.v $RTL_PATH/core.v"

report rtl_diagnostics -srclist dsgn1.pfl -top_module top

report rtl_diagnostics -vlog_std sysv -srclist mysvfiles.pfl

report rtl_diagnostics -src dsgn2.vhd
   -include_path {./path1;./path2}

set INCLUDE_PATH {./path1;./path2;./path3}
report rtl_diagnostics -src dsgn2.vhd -include_path $INCLUDE_PATH

report rtl_diagnostics -src prep2_2.v -out nocdc
```

# report slr_assignment

Run the report slr_assignment to check SLR assignments. See *Running System Route for the Top Level*, on page 412 in the *User Guide*.

## Syntax

**report slr_assignment** [**-out** *subStateName*] [**-pcf** *pcfFileNames*] [**-pcflist** *pcfListFiles*] [**-tss** *tssFileName*]

**-out** *subStateName*
Output substate name. The report name is formed by combining the name of the parent state with the sub-state name. For example *parentStateName.subStateName.*

**-pcf** *pcfFileNames*
One or more partition constraint file (PCF) files.

**-pcflist** *pcfListFiles*
A file listing one or more PCF files.

**-tss** *tssFileName*
Target system specification file.

---

**Note:** For design flows with UC 2.0 version, if the slr_assignment.pcf file is not generated by default, use the export file command to export the specified file from the current database state to the indicated directory and generate the .pcf file. For more information, see the export command.

---

# report state_diff

Generates a report that contains detailed information about the differences between two states including input files, global options, or arguments that have changed.

## Syntax

**report state_diff** *state1 state2* [**-mode current**|**active_tree**][**-out** *subStateName*]

**-mode current**|**active_tree**
: Report differences for single states or the entire sequence of states leading to them. The default is active_tree.

**-out** *subStateName*
: Outputs the sub-state name. The report name is formed by combining the name of the parent state with the sub-state name. For example *parentStateName*.*subStateName*.

# report syntax_check

Runs the HDL syntax checker on the HDL design files and reports status in the compiler log file. The command is run on the root database and automatically displays the report when complete.

## Syntax

**report syntax_check**
[**-bg**]
[**-cdc** *cdcFilename*|*tclListofCDCfiles*]
[**-cdclist** *cdcListFilename*]
[**-hdl_define** *variable*[**=***value*]]
[**-hdl_param** *parameter*=*value*]
[**-i**|**-include_path {***listofPaths***}**|*pathVariable*]
[**-ilist** *filename*]
[**-l**|**-library_path {***listofPaths***}**]
[**-lib** *libName*]
[**-llist** *filename*]
[**-lmf** *tclListofLibMapFilenames*]
[**-out** *subStateName*]
[**-src** *srcFilename*|*tclListofSourceFiles*]
[**-srclist** *srcListFilename*]
[**-srcstate** *databaseState*]
[**-top_module** *moduleName*]
[**-type** *fileType*]
[**-vlog_std** *standard*]

**-bg**

Starts the syntax checker and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-cdc** *cdcFilename*|*tclListofCDCfiles*]

Specifies the name of a compiler directives constraints file to be compiled with the design. The *cdcFilename* argument is a string value that specifies the path name to the constraint file; the *tclListofCDCfiles* argument is a standard Tcl list of one or more constraint files. If the -cdclist argument is also used, *cdcFilename* is automatically included with the other listed files

**-cdclist** *cdcListFilename*
Lists the required compiler directives constraints files in a single file (*cdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding CDC file. A filename extension is not required for *cdcListFilename*, but the name of the file must be unique.

**-hdl_define** *variable*[**=***value*]
Specifies a Tcl list of HDL defines to be applied to the top-level module.

**-hdl_param** *parameter***=***value*
Specifies a Tcl list of parameters to be applied to the top-level module. You cannot specify a file list with this argument. To specify a list of parameters, either create a file and use an include statement to add the file in the RTL code, or enclose a list of parameters in curly braces as shown below:

```
-hdl_param {W=4, K=5, X=8}
```

**-i|-include_path {***listofPaths***}|***pathVariable*
Defines the search path used for ‘include commands relative to the active directory. The argument ***listofPaths*** is a semicolon separated list of the include-file directories enclosed in curly brackets; the ***pathVariable*** is a previously-defined variable for the include paths that you can now include on the command line (see *Examples*, on page 125). The software searches for include files first in the source file directory and then in included-path directory order, stopping at the first occurrence of the include file.

**-ilist** *filename*
A file listing one or more include paths.

**-l|-library_path {***listofPaths***}|***pathVariable*
Defines the search path used for libraries relative to the active directory. The argument *listofPaths* is a semicolon separated list of the library directories or files enclosed in curly brackets (supporting a single library file format allows custom library definitions for VCS file compatibility). The *pathVariable* is a previously-defined variable for the library paths that you can now include on the command line.

**-lib** *libName*
Specifies the library to use for subsequent files (default is work). The *libName* argument is a string value that specifies the full path name to the indicated library.

**-llist**
A file listing one or more library include paths.

**-lmf** *tclListofLibMapFilenames*
Specifies a Tcl list of library mapping files. For more information on library mapping files, see Using UUM and Group Mapping, on page 109 in the *User Guide*.

**-out** *subStateName*
Specifies a text string to allow multiple reports to coexist within the same database state. The report name is formed by combining the name of the database with *subStateName*.

**-src** *srcFilename|tclListofSourceFiles*
Specifies a single HDL (Verilog or VHDL) input file or a Tcl list of source files. The *srcFilename* argument is a string value that specifies the path name to the source input file; the *tclListofSourceFiles* argument is a standard Tcl list of one or more source files. If the -srclist argument is also used, *srcFilename* is automatically included with the other listed files.

**-srclist** *srcListFilename*
Lists the required HDL (Verilog/VHDL) or edif input source files in a single file (*srcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding input file. When creating a list of source files, the top-level file either must be the last entry in the list of files or explicitly specified with a -top_module argument. A filename extension is not required for *srcListFilename*, but the name of the file must be unique.

**-srcstate** *databaseState*
Specifies the database states to be compiled in a bottom-up compile flow. The argument is repeated to identify each state to be included. To specify a path to a database state, specify the path and then use a pipe (|) between the path and the name of the database state. See *Running Bottom-Up Compile*, on page 90 in the *User Guide* for a step-by-step procedure.

**-top_module** *moduleName*
Specifies the name of the top module for the design. The *moduleName* string defines the base name for the design output files.

**-type** *fileType*
Specifies the file type applied to subsequent files when the file type is not recognized as any of the predefined file types. The *fileType* argument is a

string value; the period extension separator is assumed. The default *fileType* for NGC files is xilinx, and the default *fileType* for EDIF files is edif.

**-vlog_std** *standard*
Specifies the Verilog standard applied to subsequent Verilog files (the -vlog_std argument must precede any -src or -srclist argument). Accepted string values for *standard* are v95 (Verilog 95), v2001 (Verilog 2001), and sysv (SystemVerilog). The VHDL standard is set with the VHDL2008 option (see vhdl2008).

## Report Log

The report syntax_check command is only valid from root and produces a single report file (syntax.log). The report displayed in the Report View is named Compile Log.

## Examples

```
report syntax_check -src prep2_2.v -cdc myconsfile.cdc

report syntax_check -src "$RTL_PATH/core.v $RTL_PATH/core.v"

report syntax_check -srclist dsgn1.pfl -top_module top

report syntax_check -vlog_std sysv -srclist mysvfiles.pfl

report syntax_check -src dsgn2.vhd
   -include_path {./path1;./path2}

set INCLUDE_PATH {./path1;./path2;./path3}
report syntax_check -src dsgn2.vhd -include_path $INCLUDE_PATH
```

# report target_system

Reports the target board system configuration. Results of running the command are included in a set of report and log files that can be viewed with the view report command.

## Syntax

**report target_system**
[**-out** *subStateName*]
**-tss** *value*

**-out** *subStateName*
Specifies a text string to allow multiple reports to coexist within the same database state. The report name is formed by combining the name of the database with *subStateName*.

**-tss** *value*
Required argument. Target system specification file.

## Report Log

The report target_system command is valid from any database state and generates the following report and log files describing the target system:

- Target system specification log
- Target system specification report
- Aptn (autopartitioner) Log
- Partition & Route Target System Report

For more information on the above reports, see Chapter 5, *Inputs, Reports and Log Files* in the *ProtoCompiler Reference Manual*. Also see *Exploring Partition Choices with a Basic TSS*, on page 185 and *Defining the Target System in a Detailed TSS File*, on page 189 in the *User Guide*.

# report timing

Reports timing by running timing analysis on a mapped or system generate database state (report timing -generate) or lists which timing analysis netlist types can be run from the current database state (report timing -list).

Results of running timing analysis are included in a report file identified by this string:

*databaseStateName*.*subStateName*.*version*.ta.rpt

## Syntax for report timing

**report timing -list** [**-verbose**]

**report timing**
[**-bg**]
[**-fdc** *fileName*.fdc**]**
**[-fdclist** *fileList***]**
[*-genArgs*]
**-generate**
**-list** [**-verbose**]
[**-mode default**|**ba**]
[**-out** *subStateName*]

**-bg**
Starts timing analysis and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-fdc** *fileName*.fdc

**-fdclist** *fileList*

**-generate**
Runs timing analysis on the indicated netlist.

*-genArgs*
Specifies one of a number of timing-generation specific arguments as outlined in the following table. If no arguments are specified with the

-generate argument, the worst-case path for each clock group is reported. You can use the Tcl find command to customize the timing analysis report (see *Tcl find Command*, on page 372).

| | |
|---|---|
| **-fall_from** *clock* | Reports paths from the falling edge of the specified clock. For a given clock, selects the starting points for paths clocked on the falling edge of the clock source. The object name must be preceded by the appropriate prefix (for example, c for clock). |
| **-fall_to** *clock* | Reports paths from the falling edge of the specified clock. For a given clock, selects the ending points for paths clocked on the falling edge of the clock source. The object name must be preceded by the appropriate prefix (for example, c for clock). |
| **-from** *list* | Reports paths from the specified port, register, register pin, or clock. |
| **-max_paths** *integer* | Reports the number of paths to report for the path group. The default integer value is 1. |
| **-rise_from** *clock* | Reports paths from the rising edge of the specified clock. For a given clock, selects the starting points for paths clocked on the rising edge of the clock source. The object name must be preceded by the appropriate prefix (for example, c for clock). |
| **-rise_to** *clock* | Reports paths from the rising edge of the specified clock. For a given clock, selects the ending points for paths clocked on the rising edge of the clock source. The object name must be preceded by the appropriate prefix (for example, c for clock). |
| **-slack_margin** *floatPt* | Reports paths for the specified slack margin, which allows you to specify a floating point value for the range of worst slack times. |
| **-through** *instance* | Reports paths that pass through the specified pins or nets. |
| **-to** *list* | Reports paths to the specified ports, register, register pin, or clock. |

**-list** [**-verbose**]

Lists the types of netlists in the current state that can be used for timing analysis. Including the -verbose argument displays a short description of each netlist type following its name.

**-mode default|ba|slta**

Specifies the type of netlist for timing analysis as reported by the -list argument. These are the valid modes:

– default
  Post-synthesis netlist in the mapped database state or a post-partitioning netlist of the design as an estimate of overall system timing from the system-generate database state.

ba

Back-annotated netlist in the mapped database state.**-out** *subStateName*
Specifies a text string identifier to replace the default (tim) *subStateName* string in the timing report. The report name is formed by combining the name of the database state (for example, m0), the *subStateName,* a .ta suffix, and an .rpt extension.

For example, the string sg12.trial4.ta.rpt indicates substate trial4 for system-generate database state sg12.

## Examples

```
report timing -list -verbose

report timing -generate -netlist ba -out trial2 -max_paths 8

report timing -generate -rise_from {c:clk} -rise_to {c:clk}
```

## Report File

The report timing command is only valid from the map or system-generate database state and produces a single report file (default name *stateName*.**tim***n*.**ta**.**rpt**) that is automatically displayed. For information on analyzing this report, see *Checking Timing Results*, on page 528 in the *User Guide*.

# report vc_static

Runs the VC Static tool to analyze the design.

**report vc_static -utf** *file* **-path** *path* **-config** *file* **-show_report 1 | 0**

**-utf** *file*
Specifies the UTF file to use to run the VC Static tool.

**-path** *path*
Relative or absolute path to the output directory, for analysis results and logs.

**-config** *file*
Specifies the VC Static configuration file, which is copied from the *install_dir*/lib/spyglass directoryto the working directory

**-show_report 1 | 0**
Displays log files after analysis. 1 is the default and displays the log file; 0 does not display the log files.

# rerun

Regenerates the current state using the original input state and command.

## Syntax

**rerun** [**-out** *stateName*]

**-out** *stateName*
Sends the results of command execution to the specified database state.

# run compile

Runs the compile process of logic synthesis to translate the design to the target technology.

For information about using this command, see Compiling the Design, on page 77 in the *User Guide*.

## Syntax

**run compile**
[**-bg**]
[**-cdc** *cdcFilename*|*tclListofConstraintFiles*]
[**-cdclist** *cdcListFilename*]
[**-compile_only 1|0**]
[**-debug 0|1|2**]
[**-hdl_define** *variable*[**=***value*]]
[**-hdl_param** *parameter*=*value*]
[**-i**|**-include_path {***listofPaths***}**|*pathVariable*]
[**-idc** *idcFilename*|*TclListofIDCfiles*]
[**-idclist** *idcListFilename*]
[**-ilist** *filename*]
[**-incremental**]
[**-l**|**-library_path {***listofPaths***}**]
[**-lib** *libName*]
[**-llist** *filename*]
[**-lmf** *tclListofLibMapFilenames*]
**-nle** *netEditFilename*
[**-out** *stateName*]
[**-src** *srcFlename*|*tclListofSourceFiles*]
[**-srclist** *srcListFilename*]
[**-srcstate** *databaseState*
[**-top_module** *moduleName*]
[**-type** *fileType*]
[**-ucdb** *pathName*]
[**-upf** *upfFilename*|*tclListofUPFfiles*]
[**-upflist** *upfListFilename*]
[**-vlog_std** *standard*]

**-bg**

Starts the compilation process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-cdc** *cdcFilename|tclListofCDCfiles*

Specifies a single compiler directives constraints file or a Tcl list of constraint files. The *cdcFilename* argument is a string value that specifies the path name to the CDC file; the *tclListofCDCfiles* argument is a standard Tcl list of one or more constraint files. If the -cdclist argument is also used, *cdcFilename* is automatically included with the other listed files.

**-cdclist** *cdcListFilename*

Lists multiple compiler directives constraints files in a single file (*cdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding CDC file. A filename extension is not required for *cdcListFilename*, but the name of the file must be unique.

**-compile_only 1 | 0**

Restricts run compile to compilation and linking. The default is **0**. When set to 1, the state is flagged as *terminal* and restricted. Terminal states cannot be used for run pre_map or run pre_partition. To remove the terminal flag, rerun run compile without -compile_only 1.

The benefit of this argument is that it can be used to iterate for quick compiles and to see where RTL constraints need to be applied before some cross-boundary optimizations, such as constant propagation, are done.

When using the bottom-up compile flow, the terminal state netlist is used for combining with other blocks. The global_optimization step is done only on the final netlist. To save time, when doing bottom-up compiles, compile sub-blocks using -compile_only 1.

**-debug 0 | 1 | 2**

Determines whether a debug database is generated. If it is generated, also determines whether it is generated serially or in parallel with the compiled database.

| | |
|---|---|
| No -debug | Default. Generates debug database in sequential mode. |
| -debug 0 | Override setting that specifies that no debug database be generated.<br>You must first disable the following options to disable debug database generation using -debug 0:<br>• debug_dumpvars<br>• debug_sva<br>• verdi_mode<br>• incremental_debug<br>Results are similar to the default in most cases; the exception is described below. |
| -debug 1 | Generates the debug database sequentially, before generating the compiled database. Sequential mode is more time-intensive. Also, errors in debug database generation stop the process and the compiled database is not generated. There is no continue-on-error. |
| -debug 2 | Generates the debug database in parallel with the compiled database. This mode is faster but is more memory-intensive. |

If -debug is used in combination with a preceding run pre_instrument command, or if an idc file is specified with run compile, these factors can take precedence and change the standard behavior of the -debug settings described in the preceding table.

The table below summarizes the behavior when used with other parameters.

| -debug | run pre_instrument | idc | Effect |
|---|---|---|---|
| 0 | N | Y/N | No debug database generated. |
| 1, 2,<br>No -debug | N | Y | -debug 1 (debug database generated sequentially) |
| 0, 1, 2,<br>No -debug | Y | Y/N | Inherits debug database from pre-instrument state; no new debug database generated. |

| -debug | run pre_instrument | idc | Effect |
|---|---|---|---|
| 1 | N | N | -debug 1 (debug database generated sequentially) |
| 2 | N | N | -debug 2 (debug database generated in parallel) |
| No -debug | N | N | Default. Generates debug database in sequential mode. |

**-hdl_define** *variable*[**=***value*]
Specifies a Tcl list of HDL defines to be applied to the top-level module.

**-hdl_param** *parameter***=***value*
Specifies a Tcl list of parameters to be applied to the top-level module. You cannot specify a file list with this argument. To specify a list of parameters, either create a file and use an include statement to add the file in the RTL code or create a Tcl list of parameter assignments, for example by enclosing them in curly braces:

```
-hdl_param {W=4 K=5 X=8}
```

**-i|-include_path {***listofPaths***}|***pathVariable***
Defines the search path used for ‘include commands relative to the active directory. The argument *listofPaths* is a semicolon separated list of the include-file directories enclosed in curly brackets; the *pathVariable* is a previously-defined variable for the include paths that you can now include on the command line (see *Examples*, on page 139). The software searches for include files first in the source file directory and then in included-path directory order, stopping at the first occurrence of the include file.

**-idc** *idcFilename|tclListofIDCfiles*
Specifies a single instrumentation design constraints (IDC) file to be compiled with the design. The *tclListofIDCfiles* argument is a standard Tcl list of one or more constraint files. If the -idclist argument is also used, *idcFilename* is automatically included with the other listed files.

The IDC file manages constraints for instrumented signals. The file is automatically generated during the pre-instrumentation stage and can be further edited. It contains constraint files for the instrumented signals and break points. The file is input to the run compile command.

An essential part of this file is the Intelligent In-Circuit Emulator (IICE) which is the debug IP and establishes connections for later debug. You can have multiple IICE units. IICE parameters determine the implementation of the IICE units and configure the units so that proper communication can be established with the debugger. You can set common parameters that are shared by all the IICEs. as well as unique parameters for individual IICEs. Parameters that are common to all IICE units are set in the Instrumentor Preferences dialog box and apply to all IICE units defined for that design; the IICE parameters unique to a specific IICE definition in a multi-IICE configuration are interactively set through the tabs in the Edit IICE dialog box.

For a detailed discussion of the IDC file and IICE parameters, refer to the product documentation. Some IDC-related commands are also described in this document: *export idc*, on page 55, *iice new*, on page 88, and *trigger_group*, on page 94.

**-idclist** *idcListFilename*
: Lists the required instrumentation IDC files in a single file (*idcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding IDC list file. A filename extension is not required for *idcListFilename*, but the name of the file must be unique.

**-ilist** *filename*
: A file listing one or more include paths.

**-incremental**
: In unified compile, makes the SCM to SRS compile flow incremental. The designs or modules that are not modified are reused and only the modules which are modified for the next run gets compiled.

**-l|-library_path {***listofPaths***}|***pathVariable*
: Defines the search path used for libraries relative to the active directory. The argument *listofPaths* is a semicolon separated list of the library directories or files enclosed in curly brackets (supporting a single library file format allows custom library definitions for VCS file compatibility). The *pathVariable* is a previously-defined variable for the library paths that you can now include on the command line.

**-lib** *libName*
: Specifies the library to use for subsequent files (default is work). The *libName* argument is a string value that specifies the full path name to the indicated library.

**-llist**
: A file listing one or more library include paths.

**-lmf** *tclListofLibMapFilenames*
Specifies a Tcl list of library mapping files. For more information on library mapping files, see Using UUM and Group Mapping, on page 109 in the *User Guide*.

**-nle** *netEditFilename*
Specifies the name of the netlist editing Tcl file to be applied by the netlist editor.

**-out** *stateName*
Sends the results of the compilation to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the state is named c*n* where *n* is a sequential number.

**-src** *srcFilename|tclListofSourceFiles*
Specifies a single HDL (Verilog or VHDL) input file or a Tcl list of source files. The *srcFilename* argument is a string value that specifies the path name to the source input file; the *tclListofSourceFiles* argument is a standard Tcl list of one or more source files. If the -srclist argument is also used, *srcFilename* is automatically included with the other listed files. Either a -src or -srclist argument is required unless specified with a run pre_instrument command.

**-srclist** *srcListFilename*
Lists the required HDL (Verilog/VHDL) or edif input source files in a single file (*srcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding input file. When creating a list of source files, the top-level file either must be the last entry in the list of files or explicitly specified with a -top_module argument. A filename extension is not required for *srcListFilename*, but the name of the file must be unique.

**-srcstate** *databaseState*
Specifies the database states to be compiled in a bottom-up compile flow. The argument is repeated to identify each state to be included. To specify a path to a database state, specify the path and then use a pipe (|) between the path and the name of the database state. This example specifies two compiled states (c1 and c2) stitched into a third compiled state (a0):

```
run compile -srcstate ip1|c1 -srcstate ip2|c2 -srcstate acme|a0
   -top_module top -out top_bu_stitched
```

See *Running Bottom-Up Compile*, on page 90 in the *User Guide* for a step-by-step procedure.

**-top_module** *moduleName*
Specifies the name of the top module for the design. The *moduleName* string defines the base name for the design output files.

**-type** *fileType*
Specifies the file type applied to subsequent files when the file suffix does not indicate a predefined file type. The *fileType* argument is a string value; the period extension separator is assumed. The default *fileType* for NGC files is xilinx, and the default *fileType* for EDIF files is edif.

**-ucdb** *pathName*]
The path to the unified compile database to be compiled. When the -ucdb argument is used, the top-level module and the Verilog standard are taken from the Unified Compile database, and the -vlog_std and -top_module arguments will be ignored.

**-upf** *upfFilename*|*tclListofUPFfiles*
Specifies a single Unified Power Format (UPF) file or a Tcl list of UPF files containing the UPF commands to be compiled with the design. The *upfFilename* argument is a string value that specifies the path name to the UPF file; the *tclListofUPFfiles* argument is a standard Tcl list of one or more UPF files. If the -upflist argument is also used, *upfFilename* is automatically included with the other listed files.

**-upflist** *upfListFilename*
Lists the required Unified Power Format (UPF) files in a single file (*upfListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding UPF file. A filename extension is not required for *upfListFilename*, but the name of the file must be unique.

**-vlog_std** *standard*
Specifies the Verilog standard applied to subsequent Verilog files (the -vlog_std argument must precede any -src or -srclist argument). Accepted string values for *standard* are v95 (Verilog 95), v2001 (Verilog 2001), and sysv (SystemVerilog). The VHDL standard is set with the VHDL2008 option (see vhdl2008).

## Description

Compiles the HDL source files to create the required design files. Option set commands, entered prior to running the compile process, define the compiler mode, and the command accepts an optional compiler directives file. When the current state is root, the HDL source files are specified with either the -file or -filelist argument. When the current state is pre-instrument, no RTL files are

required to be passed to the process (the design data is inherent in the database state). Successful execution of the run compile command results in the creation of a new database state.

## Report Log

The run compile command produces a single report file (*designName*_compile.srr) which can be displayed by a view report command entered from the compile database (or subsequent) state. The displayed report in the Report View is named Compile Log.

## Termination Status

Execution of the run compile command returns:

0 – no error

2 – errors (see the log file)

## Examples

The following examples illustrate a number of the run compile command options:

### Examples of adding RTL files

```
run compile -src prep2_2.v -cdc myconsfile.cdc -out CMPstate1

run compile -src "$RTL_PATH/core.v $RTL_PATH/core.v"

run compile -srclist dsgn1.pfl -top_module top
```

### Examples of using the library mapping file option

```
run compile -lmf ./vloglib.map

run compile -lmf "./vloglib1.map;./vloglib2.map"
```

### Examples of using the -include_path option

```
run compile -src dsgn2.vhd -include_path {./path1;./path2}

set INCLUDE_PATH {./path1;./path2;./path3}
run compile -src dsgn2.vhd -include_path $INCLUDE_PATH
```

### Example of using the -hdl_define option

```
run compile -srclist add_files.txt -hdl_define {M1=2 M2 M3=3}
```

### Example of using the -vlog_std option

```
run compile -vlog_std sysv -srclist mysvfiles.pfl
```

### Example of using the -incremental option

```
launch uc -utf run.utf -ucdb ucdb
run compile -ucdb ucdb -out c0 -incremental
```

# run debug_eco

After creating an incremental instrumentation file from a map state with edit idc, use the run debug_eco command to generate the script to implement these changes in the post place-and-route netlist. Then use export vivado and launch vivado to execute the ECO script. Instrumentation changes made in this way are limited, but can be made very quickly as they occur on the post place-and-route netlist.

The run debug_eco command creates a new database substate under the parent map state. This substate is a terminal state and does not generate any other database states. For information on the ECO debug flow, see Running Incremental Debug for ECOs, on page 748 in the *User Guide*.

## Syntax

**run debug_eco -idc** *idcFilename* [**-bg**] [**-out** *stateName*]

**-idc** *idcFilename*
The name of the incremental IDC file.

**-bg**
Starts command execution and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-out** *stateName*
Sends the results of the compilation to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the state is named de*n* where *n* is a sequential number.

# run map

Runs the mapping process of logic synthesis to translate the design to the targeted technology. The command is available after successful execution of the pre-map task.

For information about using this command, see Mapping the Design, on page 490 in the *User Guide*.

## Syntax

**run map**
[**-bg**]
[**-out** *stateName*]

**-bg**
Starts the mapping process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-out** *stateName*
Sends the results of mapping to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named m*n* where *n* is a sequential number.

## Report Log

The run map command produces a single map log file (map.srr) that can be displayed by a view report command entered from the map database (or subsequent) state.

# run par_explorer

Performs place and route exploration. Use option set max_parallel_par_explorer to define the maximum number of concurrent place and route jobs.

For information about using this command, see *Running Place-and-Route Exploration*, on page 576 in the *User Guide*.

## Syntax

**run par_explorer** [**-bg**] [**-out** *stateName*] [**-script** *scriptName*]

**-bg**
Starts the place-and-route explore process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-out** *stateName*
Sends the results of place-and-route explore to the specified database state. If the state does not exist, it is created, an if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named pe*n* where *n* is a sequential number.

**-script** *scriptName*
Specifies a custom script to use with exploratory place and route. The script must be in the same format as the run_haps_vivado.tcl script; copy run_haps_vivado.tcl, make the necessary edits, then save it with a new name (*scriptName*).

## Report Log

The run par_explorer command produces a single log file (proto_haps_dx_top.srr) that can be displayed by a view report command entered from the par-explorer database state.

# run partition

Generates a partitioned netlist using PCF constraints. The command is available after successful execution of the pre-partition task.

For information about using this command, see Partitioning the Logic, on page 328 in the *User Guide*.

## Syntax

**run partition**
  [**-bg**]
  **-builtin_est 0 | 1**
  [**-clock_gate_replication 0|1**]
  [**-continue_on_pcf_error 0|1**]
[**-effort low|normal|high**]
  [**-mode normal|preassigned|auto_hierarchical|manual_hierarchical**]
  [**-multi_thread 0|1**]
  [**-optimization_priority tdm_ratio|nets|multi_hop_path|preassigned**]
  [**-out** *stateName*]
  [**-pcf** *filename*|*tclListofPCFfiles*]
  [**-pcflist** *pcfListFilename*]
  [**-tss** *sysSpecFile*.**tcl**]
(The following advanced option is used to guide partition exploration)
[**-pmux_decomposition 0|1**]
  [**-builtin_est 0|1**]

**-bg**
Starts the partitioning process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-builtin_est 0 | 1**
Uses the less accurate built-in area estimation data instead of using any area estimation generated during pre_partition or imported via database apply_state -import_est. Recommended for use only when run prepartition -area_est is not used. Default is 0.

**-clock_gate_replication 0|1**
Automatically replicates clock trees and requisite gates in multi-FPGA designs to eliminate the need for manual clock replication; default is 0 (disable replication). Setting the argument to 1 automatically replicates the entire clock tree into the FPGAs, reports the replicated clock nets in

the partition.log file, and generates a PCF file (auto_replication.pcf) of the corresponding PCF commands. The *clock_gate_replication_control* PCF command allows clock replication logic to be assigned to a locked bin.

**-continue_on_pcf_error 0|1**
Prints only a warning and continues if the object in the PCF file is not present in the design.

**-effort low|normal|high**
The -effort option is equivalent to using several options to set values as shown in the following table.

| Option | Low | Normal | High |
|---|---|---|---|
| **-max_trials** | 150 | 1000 | 1000 |
| **-solutions** | 1 | 3 | 6 |
| **-hierarchy_dissolve_pass** | 1 | 2 | 2 |
| **-multi_hop_accuracy** | 0 (estimate) | 0 (estimate) | 1 (full accuracy) |

The default -effort is normal. Subsequent use of the individual options overrides the values set using -effort.

**--mode normal|preassigned|auto_hierarchical|manual_hierarchical**
Sets the mode to use for partitioning. The default mode is normal. The auto_hierarchical and manual_hierarchical (default) mode options specify automatic and manual hierarchical partitioning, respectively. See *Using Hierarchical Partitioning*, on page 406 in the *User Guide* for usage information. With preassigned mode, the partitioner uses the constraints in assignments.pcf as a starting point.

**-multi_thread 0|1**
Allows the partitioner to use up to four threads. Setting -multi_thread to 0 forces the partitioner to use only one thread. The default is 1.

**-optimization_priority tdm_ratio|nets|multi_hop_path|preassigned**
Sets the objective for the system router. Specifying tdm_ratio (the default) attempts to reduce the maximum TDM ratio and nets attempts to reduce the number of nets. Specifying multi_hop_path attempts to reduce the number of combinational paths that cross between FPGAs. With preassigned, the partitioner uses the constraints in assignments.pcf as a starting point.

**-out** *stateName*
: Sends the results of partitioning to the specified database state. If the state does not exist, it is created, an if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named pa*n* where *n* is a sequential number.

**-pcf** *pcfFilename|tclListofPCFfiles*
: Specifies a single partition constraints file (PCF) or a Tcl list of constraint files that define the partitioning parameters. The *pcfFilename* argument is a string value that specifies the path name to the PCF file; the *tclListof-PCFiles* argument is a standard Tcl list of one or more constraint files. If the -pcflist argument is also used, *srcFilename* is automatically included with the other listed files.

**-pcflist** *pcfListFilename*
: Lists the required PCF files in a single file (*pcfListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding PCF file. A filename extension is not required for *pcfListFilename*, but the name of the file must be unique.

**-tss** *sysSpecFile*.**tcl**
: Overrides the target physical system specification used in the pre-partition process.

## Exit Status

The auto_partitioner job can exit with one of the status codes shown in the table below, or a different code determined by your operating system:

| Status | Description |
|---|---|
| 0 | Successful execution |
| 2 | Error in autopartitioner; see error message in partition.log (or route.log) |
| 20 | Error in backend; error message in partition_build.log |
| 100 | Error in autopartitioner during feedthrough insertion or net splitting; see error message in route.log |

## Report Logs

The run partition command produces a set of report log files that can be displayed by a view report command entered from the partition (or subsequent) database state. Individual partitioning reports can be explicitly displayed by entering the name string argument for the view report command as shown in the following table:

| view report Command Argument | Report Name in Report View |
|---|---|
| partition.log | Partition Log |
| partition.rpt | Partition Report |
| timing.rpt | Multi-hop Path Report |
| partition_build.log | Partitioned netlist creation log |
| tdm_nonqualified.rpt | Report of all nets disqualified for TDM |
| tdm_qualified.rpt | Report of all nets qualified for TDM |

The -beta option adds the following additional sections to the partition log:

| Section | Message ID | Title |
|---|---|---|
| S3.2.2 | AP507 | Initializing Timing Graph |
| S3.2.3 | AP579 | Timing Graph Clock Summary |
| S4.7 | AP431 | Clock Summary |
| S4.8 | AP431 | Critical Path Summary |

# run postpar_resynthesis

Runs post-partition resynthesis on the existing mapped database using the specified Xilinx DCP file for backannotation. To use this command, the Vivado database must be placed or routed. Resynthesis is performed to improve timing QoR and routability of the netlist.

## Syntax

**run postpar_resynthesis -dcp** *filename* [**-bg**] [**-out** *stateName*]

**-dcp** *filename*
Specifies the name of the Xilinx DCP file for backannotation.

**-bg**
Starts the post-partition resynthesis process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-out** *stateName*
Sends the results of resynthesis to the specified database state. If the state does not exist, it is created, an if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named pr*n* where *n* is a sequential number.

# run pre_instrument

Runs the pre-instrument design process. The command creates a debug database for the design source to allow debug instrumentation of RTL through an IDC constraint file during the compile, pre-partition, or pre-map state. The command is available from root and is not required for netlist-instrumentation.

For information about using this command, see *Instrumenting the Design for Debug*, on page 641 in the *User Guide*.

## Syntax

**run pre_instrument**
[**-bg**]
[**-cdc** *cdcFilename*|*tclListofCDCfiles*]
[**-cdclist** *cdcListFilename*]
[**-hdl_define** *variable*[**=***value*]]
[**-hdl_param** *parameter*=*value*]
[**-i**|**-include_path {***listofPaths***}**|*pathVariable*]
[**-ilist** *filename*]

[**-l**|**-library_path {***listofPaths***}**]
[**-lib** *libName*]
[**-llist** *filename*]
[**-lmf** *tclListofLibMapFilenames*]
[**-out** *stateName*]
[**-src** *srcFilename*|*tclListofSourceFiles*]
[**-srclist** *srcListFilename*]
[**-srcstate** *databaseState*]
[**-top_module** *moduleName*]
[**-type** *fileType*]
[**-vlog_std** *standard*]

**-bg**
Starts the pre-instrument process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-cdc** *cdcFilename*|*tclListofCDCfiles*
Specifies a single compiler directives constraints file or a Tcl list of constraint files. The *cdcFilename* argument is a string value that speci-

fies the path name to the CDC file; the *tclListofCDCfiles* argument is a standard Tcl list of one or more constraint files. If the -cdclist argument is also used, *cdcFilename* is automatically included with the other listed files.

**-cdclist** *cdcListFilename*
Lists the required compiler directives constraints files in a single file (*cdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding CDC file. A filename extension is not required for *cdcListFilename*, but the name of the file must be unique.

**-hdl_define** *variable*[**=***value*]
Specifies a Tcl list of HDL defines to be applied to the top-level module.

**-hdl_param** *parameter***=***value*
Specifies a Tcl list of parameters to be applied to the top-level module. You cannot specify a file list with this argument. To specify a list of parameters, either create a file and use an include statement to add the file in the RTL code, or enclose a list of parameters in curly braces as shown below:

```
-hdl_param {W=4, K=5, X=8}
```

**-i|-include_path {***listofPaths***}|***pathVariable*
Defines the search path used for 'include commands relative to the active directory. The argument *listofPaths* is a semicolon separated list of the include-file directories enclosed in curly brackets; the *pathVariable* is a previously-defined variable for the include paths that you can now include on the command line (see *Examples*, on page 152). The software searches for include files first in the source file directory and then in included-path directory order, stopping at the first occurrence of the include file.

**-ilist** *filename*
A file listing one or more include paths.

**-l|-library_path {***listofPaths***}|***pathVariable*
Defines the search path used for libraries relative to the active directory. The argument *listofPaths* is a semicolon separated list of the library directories or files enclosed in curly brackets (supporting a single library file format allows custom library definitions for VCS file compatibility). The *pathVariable* is a previously-defined variable for the library paths that you can now include on the command line.

**-lib** *libName*
Specifies the library to use for subsequent files (default is work). The *libName* argument is a string value that specifies the full path name to the indicated library.

**-llist**
A file listing one or more library include paths.

**-lmf** *tclListofLibMapFilenames*
Specifies a Tcl list of library mapping files. For more information on library mapping files, see Using UUM and Group Mapping, on page 109 in the *User Guide*.

**-out** *stateName*
Sends the results of the compilation to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named r*n* where *n* is a sequential number.

**-src** *srcFilename|tclListofSourceFiles*
Specifies a single HDL (Verilog or VHDL) input file or a Tcl list of source files. The *srcFilename* argument is a string value that specifies the path name to the source input file; the *tclListofSourceFiles* argument is a standard Tcl list of one or more source files. If the -srclist argument is also used, *srcFilename* is automatically included with the other listed files. Either a -src or -srclist argument is required unless specified with a run compile command.

**-srclist** *srcListFilename*
Lists the required HDL (Verilog/VHDL) or edif input source files in a single file (*srcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding input file. When creating a list of source files, the top-level file either must be the last entry in the list of files or explicitly specified with a -top_module argument. A filename extension is not required for *srcList-Filename*, but the name of the file must be unique.

**-srcstate** *databaseState*
Specifies the database states to be compiled in a bottom-up compile flow. The argument is repeated to identify each state to be included. To specify a path to a database state, specify the path and then use a pipe (|) between the path and the name of the database state. This example specifies two compiled states (c1 and c2) stitched into a third compiled state (a0):

```
run pre_instrument -srcstate ip1|c1 -srcstate ip2|c2
   -srcstate acme|a0 -top_module top -out top_bu_stitched
```

See *Running Bottom-Up Compile*, on page 90 in the *User Guide* for a step-by-step procedure.

**-top_module** *moduleName*
Specifies the name of the top module for the design. The *moduleName* string defines the base name for the design output files.

**-type** *fileType*
Specifies the file type applied to subsequent files when the file type is not recognized as any of the predefined file types. The *fileType* argument is a string value; the period extension separator is assumed. The default *fileType* for NGC files is xilinx, and the default *fileType* for EDIF files is edif.

**-vlog_std** *standard*
Specifies the Verilog standard applied to subsequent Verilog files (the -vlog_std argument must precede any -src or -srclist argument). Accepted string values for *standard* are v95 (Verilog 95), v2001 (Verilog 2001), and sysv (SystemVerilog). The VHDL standard is set with the VHDL2008 option (see vhdl2008).

## Report Log

The run pre_instrument command produces a single log file (*designName*_identify_db_generator.srr) that can be displayed by a view report command entered from the pre-instrument (or subsequent) database state. The report displayed in the Report View is named Pre_Instrument Log.

## Examples

```
run pre_instrument -src prep2_2.v -cdc myconsfile.cdc run
pre_instrument -srclist dsgn1.pfl -top_module top

run pre_instrument -vlog_std sysv -srclist mysvfiles.pfl

run pre_instrument -src dsgn2.vhd
   -include_path {./path1;./path2}

set INCLUDE_PATH {./path1;./path2;./path3}
run pre_instrument -src dsgn2.vhd -include_path $INCLUDE_PATH
```

# run pre_map

Executes the pre-map design process. The command is available after successful execution of the compile task in the synthesis design flow.

For information about using this command, see Running Pre-Map, on page 474 in the *User Guide*.

## Syntax

**run pre_map**
[**-bg**]
[**-fdc** *fdcFilename*|*tclListofFDCfiles*]
[**-fdclist** *fdcListFilename*]
[**-idc** *idcFilename*|*tclListofIDCfiles*]
[**-idclist** *idcListFilename*]
[**-out** *stateName*]

**-bg**
Starts the pre-map process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-fdc** *fdcFilename*|*tclListofFDCfiles*
Specifies a single FPGA design constraints file or a Tcl list of constraint files to be compiled with the design. The *fdcFilename* argument is a string value that specifies the path name to the FDC file; the *tclListofFDCfiles* argument is a standard Tcl list of one or more constraint files. If the -fdclist argument is also used, *fdcFilename* is automatically included with the other listed files.

**-fdclist** *fdcListFilename*
Lists the required FPGA design constraints files in a single file (*fdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding FDC file. A filename extension is not required for *fdcListFilename*, but the name of the file must be unique.

**-idc** *idcFilename*|*tclListofIDCfiles*
Specifies a single instrumentation design constraints (IDC) file or a Tcl list of IDC files to be compiled with the design. The *idcFilename* argument is a string value that specifies the path name to the IPC file; the *tclListofIDCfiles* argument is a standard Tcl list of one or more

constraint files. If the -idcflist argument is also used, *idcFilename* is automatically included with the other listed files. The -idc option is required in the pre-map state only when performing SRS instrumentation or mixed RTL and SRS instrumentation.

**-idclist** *idcListFilename*

Lists the required instrumentation design constraints (IDC) files in a single file (*idcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding IDC list file. A filename extension is not required for *idcListFilename*, but the name of the file must be unique.

**-out** *stateName*

Sends the results of the compilation to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named pm*n* where *n* is a sequential number.

## Report Logs

The command produces two report log files that can be displayed by a view report command entered from the pre-map (or subsequent) database state. The individual reports can be explicitly displayed by entering the name string argument for the view report command as shown in the following table:

| view report Command Argument | Report Name in Report View |
|---|---|
| pre_map.srr | Pre_Map Log |
| haps_io_report.txt | HAPS IO Report |

# run pre_partition

Runs the pre-partitioning process. The command is available after successful execution of the compile task in the partition design flow.

For information about using this command, see *Using run pre_partition to Define Partitions*, on page 324 in the *User Guide*.

## Syntax

**run pre_partition**[**-out** *stateName*]
**-tss** *sysSpecFile*.**tcl**
[**-area_est 0**|**1**]
[**-bg**]
[**-fdc** *fdcFilename*|*tclListofFDCfiles*]
[**-fdclist** *fdcListFilename*]
[**-idc** *idcFilename*|*tclListofIDCfiles*]
[**-idclist** *idcListFilename*]
[**-out** *stateName*]

**-area_est 0|1**
Generates area estimation data. The tool runs pre-partition with built-in area estimation (fast area estimation) when set to **0**. To run regular area estimation, use this argument set to **1**. The default is **1**.

**-bg**
Starts the pre-partitioning process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-fdc** *fdcFilename*|*tclListofFDCfiles*
Specifies a single FPGA design constraints file or a Tcl list of constraint files to be compiled with the design. The *fdcFilename* argument is a string value that specifies the path name to the FDC file; the *tclListofFDCfiles* argument is a standard Tcl list of one or more constraint files. If the -fdclist argument is also used, *fdcFilename* is automatically included with the other listed files.

The pre-partition state is the first state where FDC constraints are honored. FDC files can be specified at later stages as well, for additional constraints.

**-fdclist** *fdcListFilename*
: Lists the required FPGA design constraints files in a single file (*fdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding FDC file. A filename extension is not required for *fdcListFilename*, but the name of the file must be unique.

**-idc** *idcFilename*|*tclListofIDCfiles*
: Specifies a single instrumentation design constraints (IDC) file to be applied during the pre-partition state for incremental update of ECO debug signals. The *tclListofIDCfiles* argument is a standard Tcl list of one or more constraint files. If the -idclist argument is also used, *idcFilename* is automatically included with the other listed files.

**-idclist** *idcListFilename*
: Lists the required instrumentation IDC files for incremental ECO debug within a single file (*idcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding IDC list file. A filename extension is not required for *idcListFilename*, but the name of the file must be unique.

**-out** *stateName*
: Sends the task results to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the state is named pp*n* where *n* is a sequential number.

**-tss** *sysSpecFile*.**tcl**
: Required argument. Specifies the target physical system specification.

## Report Logs

The command produces four report log files that can be displayed by a view report command entered from the pre-partition (or subsequent) database state. The individual reports can be explicitly displayed by entering the name string argument for the view report command as shown in the following table:

| view report Command Argument | Report Name in Report View |
|---|---|
| netlist_optimizer.log | Netlist Editing and Optimization Log |
| tss.log | Target system specification log |
| pre_map.log | Pre-partition Log |
| area_estimation.log | Area Estimation Log |

# run system_generate

Runs the system-generate process, which creates a timing budget and generates partitions based on the hardware specification in the tss file and the pcf and fdc constraints.

Note that when single-FPGA synthesis scripts and databases are generated, only options that have non-default values are forwarded from the multi-fpga database to the single-fpga database.

For information about using this command, see Generating FPGAs, on page 416 in the *User Guide*.

## Syntax

**run system_generate**
**-path** *pathName*
[**-bg**]
[**-database 0|1**]
[**-fdc** *fdcFilename|tclListofFDCfiles*]
[**-fdclist** *fdcListFilename*]
[**-idc** *idcFileName|tclListofIDCfiles*]
[**-idclist** *idcListFilename*]
[**-out** *stateName*]
[**-subsystem** *subsystemName*]
[**-synthesis_template**]
[**-time_budget 0|1**]
[**-verilog 0|1**]

**-path** *pathName*
A required argument that specifies a directory outside the database where single-FPGA databases will be written. The database for each FPGA will be written to a separate directory in that location, named after the FPGA.

**-bg**
Starts the system-generate process and immediately returns the shell prompt. With the **-bg** argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-database 0|1**
Generates SRS files for single FPGA projects; default is 1.

**-fdc** *fdcFilename|tclListofFDCfiles*

Specifies a single FPGA design constraints file or a Tcl list of constraint files to be compiled with the design. The *fdcFilename* argument is a string value that specifies the path name to the FDC file; the *tclListofFDCfiles* argument argument is a standard Tcl list of one or more constraint files. If the -fdclist argument is also used, *fdcFilename* is automatically included with the other listed files.

This file is first specified at the pre-partition stage, but must be specified again at this stage, so that the generated partitions are constrained properly. Some constraints only take effect at this stage.

**-fdclist** *fdcListFilename*

Lists the required FPGA design constraints files in a single file (*fdcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding FDC file. A filename extension is not required for *fdcListFilename*, but the name of the file must be unique.

**-idc** *idcFilename|tclListofIDCfiles*

Specifies a single instrumentation design constraints (IDC) file to be applied during the system-generate state for incremental update of ECO debug signals. The *tclListofIDCfiles* argument is a standard Tcl list of one or more constraint files. If the -idclist argument is also used, *idcFilename* is automatically included with the other listed files.

**-idclist** *idcListFilename*

Lists the required instrumentation IDC files for incremental ECO debug within a single file (*idcListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding IDC list file. A filename extension is not required for *idcListFilename*, but the name of the file must be unique.

**-out** *stateName*

Sends the results of system generate to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the database state is named sg*n* where *n* is a sequential number.

**-subsystem** *subsystemName*

Quickly generates a new SLP database for the specified subsystem. This option is used in incremental runs. *subsystemName* must match the name specified with the subsystem command in PCF.

**-time_budget 0|1**
Generates time budgets for single FPGA projects by default; 0 disables time-budget generation.

**--synthesis_template**
Path to Tcl file to be used for synthesis. Default is $LIB/protocompiler/sysgen_template.tcl.

**-verilog 0|1**
Generates Verilog for single FPGA projects; default is 1.

## Report Logs

The command produces five report log files that can be displayed by a view report command entered from the system-generate (or subsequent) database state. The individual reports can be explicitly displayed by entering the name string argument for the view report command as shown in the following table:

| view report Command Argument | Report Name in Report View |
|---|---|
| board_iostd.rpt | Board IO STD Report |
| system_generate.log | System generate log |
| pre_map.srr | Pre-map Log |
| *designName*_slpgen.srr | Syntax Constraint Check Report |
| time_budget.log | Time Budgets Generation Log |

# run system_route

Runs the FPGA route process by inserting TDM logic on the partitioned design and then routing all inter-FPGA signals to system traces. The command is available after successful execution of the partition task, and cannot be used with abstract_tss commands.

For information about using this command, see *Running System Route for the Top Level*, on page 412 in the *User Guide*.

## Syntax

**run system_route**
[**-bg**]
**-continue_on_pcf_error 0**|**1**
**-estimate_timing 0**|**1**|*fileName*
[**-fdc** *fdcFilename*|*tclListofFDCfiles*]
[**-fdclist** *fdcListFilename*
**-optimization_priority tdm_ratio|multi_hop_path|slack**
[**-out** *stateName* ]
[**-mapped_timing_models 0**|**1**]
[**-pcf** *filename*|*tclListofPCFfiles* ]
[**-pcflist** *pcfListFilename* ][**-tss** *sysSpecFile*.**tcl** ]

**-bg**
Starts the routing process and immediately returns the shell prompt. With the -bg argument, license-based jobs are still run sequentially with the next job starting only on completion of the previous job. Unlicensed or third-party jobs are run concurrently.

**-continue_on_pcf_error 0**|**1**
Prints only a warning and continues if the object in the PCF file is not present in the design.

**-estimate_timing 0**|**1**|*fileName*
When 1 is set, this argument generates an estimated timing file and uses it for routing optimization. When *fileName* is set, this argument uses the specified file. The default is **0.**

**-fdc** *fdcFilename*|*tclListofFDCfiles*
Specifies a single FPGA design constraints file or a Tcl list of constraint files to be compiled with the design. The *fdcFilename* argument is a string value that specifies the path name to the FDC file; the *tclListofFD-*

*Cfiles* argument is a standard Tcl list of one or more constraint files. If the -fdclist argument is also used, *fdcFilename* is automatically included with the other listed files.

This file is first specified at the pre-partition stage, but some constraints only take effect at this stage. Specify it again for that reason, or to add or refine constraints.

**-fdclist** *fdcListFilename*

Lists the required FPGA design constraints files in a single file (*fdcList-Filename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding FDC file. A filename extension is not required for *fdcListFilename*, but the name of the file must be unique.

**-optimization_priority tdm_ratio|multi_hop_path|slack**

Sets the objective for the system router. Specifying tdm_ratio (the default) attempts to reduce the maximum TDM ratio. Specifying multi_hop_ path attempts to reduce the number of combinational paths that cross between FPGAs. The slack objective enables top-down timing and sets the optimization priority to consider best-case slack (requires the appropriate constraints to be defined in the FDC file). To use this argument, -estimate_timing must be disabled (set to 0). See Using Timing-Aware Partitioning, on page 341 for more information.

**-out** *stateName*

Sends the routing results to the specified database state. If the state does not exist, it is created, and if the state exists, it is overwritten. The *stateName* argument is a string value. In the absence of this argument, the state is named sr*n* where *n* is a sequential number.

**-mapped_timing_models 0|1**

Generates accurate timing models from mapped netlist to improve pre-tdm and post-tdm slack correlation between system route and system generate. Default is 0.
Enabling this option improves correlation in multi-hop paths with clock groups, clock crossing paths and individual path timing exceptions. With accurate timing models, router can insert appropriate TDM ratios that improve overall design performance. When this option is enabled, the tool runs estimate timing during system route and annotates the delay in the top-down timing (TDT) graph during system route.
Use this option with -optimization_priority slack:

```
run system_route -optimization_priority slack -mapped_timing_-
models 1
```

**-pcf** *pcfFilename|tclListofPCFfiles*
Specifies a single partition constraints file (PCF) or a Tcl list of constraint files that define the partitioning parameters. The *pcfFilename* argument is a string value that specifies the path name to the PCF file; the *tclListof-PCFfiles* argument is a standard Tcl list of one or more constraint files. If the -pcflist argument is also used, *pcfFilename* is automatically included with the other listed files.

**-pcflist** *pcfListFilename*
Lists the required PCF files in a single file (*pcfListFilename*). Each entry within this file is on a separate line and each entry must specify the path name to the corresponding PCF file. A filename extension is not required for *pcfListFilename*, but the name of the file must be unique.

**-tss** *sysSpecFile*.**tcl**
Sets the target system specification file for system route and overrides the TSS file definition from the pre-partition or partition phase.

## Exit Status

Returns the status codes shown in the following table to indicate execution results.

| Status | Description |
|---|---|
| 0 | Successful execution |
| 2 | Error in autopartitioner; see error message in route.log (or partition.log) |
| 20 | Error in backend; error message in partition_build.log |
| 100 | Error in autopartitioner during feedthrough insertion or net splitting; see error message in route.log |

## Report Logs

The run system_route command produces a set of report log files that can be displayed by a view report command entered from the system-route (or subsequent) database state. Individual reports can be explicitly displayed by entering the name string argument for the view report command as listed in the following table:

| view report Command Argument | Report Name in Report View |
|---|---|
| system_route.log | System Route Log |
| system_route.rpt | System Route Report |
| timing.rpt | Multi-hop Path Report |
| route_build.log | Partitioned netlist creation log |

The -beta option adds the following additional sections to the system_route.log:

| Section | Message ID | Title |
|---|---|---|
| S3.1 | AP603 | Non-Timing-Driven Global Route |
| S2.3 | AP506 | Reading Timing Constraints |
| S3.4 | AP604 | Generating Timing Graph |
| S3.4.1 | AP126 | Dissolving Design for Timing |
| S3.4.2 | AP507 | Initializing Timing Graph |
| S5.5.3 | AP617 | Optimizing for Slack |

# view

A set of commands that provides views into the design project. These views include design schematics and database reports.

## Syntax

**view**
**formality_report -path** *reportFilepath*
**report** [*reportFileName*]
**schematic** [**-list** [**-verbose**] |**-database** *databasePath*|*netlist*]

**formality_report -path** *reportFilepath*
Opens a formal verification report from an exported location.

**report** [*reportFileName*]
Displays reports available from within the database. If only report is entered, displays all available reports tracing back from the current database state to the beginning of its database hierarchy. If *reportFile-name* is specified, opens the requested report while making all of the reports from the preceding database states available. To display the name or names of the reports available for a database state, use the report list command.

**schematic** [**-list** [**-verbose**] |**-database** *databasePath*|*netlist*]
Opens an HDL Analyst® window to display a schematic showing the current design phase. When a compile or pre-map database state is active, the RTL view is displayed, and when a map or subsequent state is active, the corresponding technology view is displayed. The -list argument displays a list of the schematics available from the current database state; including the -verbose argument prints a short description of each available netlist. Using *-database* you can designate the path to a different database than the current one. The *netlist* argument explicitly specifies the netlist to view and is mutually exclusive with the -list argument.

SYNOPSYS®
Silicon to Software™

CHAPTER 3

# Partition Constraint File Tcl Commands

This chapter describes the Tcl constraint commands entered into a partition constraint file (PCF) to control the partitioning operation. This file is read by both the run partition and run system_route commands using the -pcf option. The command descriptions in this chapter are categorized as follows:



The table below lists the PCF commands in alphabetical order.

| | | |
|---|---|---|
| abstract_tss | design_list_cells | reserve_trace |
| assign_cell | design_list_nets | reset_synchronize |
| assign_global_net | design_list_ports | |
| assign_port | design_list_properties | tdm_control |
| auto_tss_control | dissolve_control | trace_group_attribute |
| bin_attribute | hierarchical_super_bin | tss_list_bins |
| bin_utilization | net_attribute | tss_list_functions |
| cell_attribute | net_route | tss_list_pins |
| cell_utilization | partition_optimization_control | tss_list_properties |

| | | |
|---|---|---|
| clock_gate_replication_control | pcf_mode | tss_list_trace_groups |
| cluster | port_attribute | tss_list_traces |
| cluster_net | replicate_cell | zcei_assign_route |
| cluster_port | report_control | |

# Target System Definition

Tcl commands in this category modify attributes of the target system and include the following commands:

# abstract_tss

Defines the trace groups and port bins for an abstract tss file.

## Syntax

**abstract_tss**
**-clear_fpga_traces** |
**-add_port_bin -fpga** *binName* **-width** *integer* **-name** *name*
[**-locked**] [**-function** *functionName*] |
**-add_trace_group {***binList***} -width** *integer* [**-connect_all**] [**-name** *name*]
[**-function** *functionName*]

**-clear_fpga_traces**
Clears all FPGA traces.

**-add_port_bin -fpga** *binName* **-width** *integer* [**-name** *name*]
[**-locked**] [**-function** *functionName*]
Creates a new port bin connected by a single trace group to the specified FPGA bin. The width of the group is specified by the -width argument. The resulting trace group is marked with connection model DIRECT. The -name argument assigns an identifying name to the port bin. When the optional -locked attribute is included, all PCF assignments to the bin are honored, and the partitioner does not change the contents of the bin; if no assignments have been made to the bin, the bin remains empty.

**-add_trace_group {***binList***} -width** *integer* [**-connect_all**] [**-name** *name*]
[**-function** *functionName*]
Creates a new trace group made up of *binList*. The width of the group connected to the specified bins is determined by the -width argument, and only connections between FPGAs are allowed. The optional -connect_all argument creates trace groups that interconnect all pairs of the specified bins. The optional -name argument assigns an identifying name to the trace group, and the optional -function argument annotates the generated traces with *functionName*.

## Examples

```
abstract_tss -clear_fpga_traces

abstract_tss -add_trace_group {FB1.uA FB1.uB} -width 96

abstract_tss -add_port_bin -fpga FB1.uA -name PCI
   -width 200 -locked
```

# bin_attribute

Enables changing the attributes on a set of bins.

## Syntax

**bin_attribute** *binGroupName* [**-locked**]

*binGroupName*
The name of a bin group.

**-locked**
Changes the state of the -locked attribute. When the -locked attribute is specified, all PCF assignments to the bin are honored, and the partitioner does not change the contents of the bin.

## Example

```
bin_attribute g3 -locked
```

## Error

An error is reported if *binGroupName* does not exist

## Return Value

Returns the bin name if successful, else an empty list.

# tdm_control

Defines parameters for using time-domain multiplexing.

## Syntax

**tdm_control** [**-type** *moduleType*]
[**-hstdm_reset_trace** *traceName*|**none**] [**-hstdm_bit_rate** *rate*]
[**-acpm_fast_clock_trace** *traceName* [**-acpm_fast_clock_freq** *frequency*]]
[**-min_ratio** *floatingPoint*] [**-max_ratio** *floatingPoint*]
[**-min_mgtdm_ratio** *floatingPoint*] [**-max_mgtdm_ratio** *floatingPoint*]
[**-allowed_modules {***moduleList***}**]
[**-qualification_mode** {**all**|**seq2seq**|**startseq**|**endseq**}]
[**-tdm_fast_default_factor** *factor*]
[**-combine_equiv_async_tdm_traces 0**|**1**]
[**-tdm_feedthrough 0**|**1**]
[**-qualify_clk_enable_paths 0**|**1**]
[**-hstdm_auto_bitrate_selection 0**|**1**]
[**-enable_mgtdm 1**|**0**|**auto**]
[**-low_latency_mgtdm 1**]

**-acpm_fast_clock_trace** *traceName* [**-acpm_fast_clock_freq** *frequency*]
For ACPM modules type, specifies the GCLK trace name for the target system. The GCLK trace must be assigned to one of the GCLKs (the router checks that the trace is in functional group gclk). Only one fast clock is supported and all ratios use this clock (when more than one fast clock is defined, the last one encountered is used). Specifying a frequency with the -acpm_fast_clock_freq argument is optional and defaults to 80 MHz. No cross-checking of the clock frequency in the file and on the board is made to allow tighter frequencies to be set for synthesis and place-and-route. This command argument implies a timing constraint; defining a fast clock in an FDC file is not recommended.

**-allowed_modules {***moduleList***}**
Specifies a space-separated list of module types that are allowed for TDM (by default, all module types are allowed). When using this option, all required modules must be identified. The -allowed_modules option cannot be used with the -min_ratio and -max_ratio options. Including this option does not change the connection-model type.

**-combine_equiv_async_tdm_traces 0|1**

Combine all equivalent traces with ASYNC TDM connection model in the global router. The default is 0 (no combining).

**-enable_mgtdm** [**1|0|auto**]

Runs MGTDM in different modes. 1 is the default mode; runs MGTDM at 12.5 Gbps. The auto mode, runs MGTDM automatically at 12.5 Gbps and selects the TDM ratio based on the pre-tdm slack value of the inter-FPGA nets. Using mode 0 disables MGTDM.

**-hstdm_auto_bitrate_selection 0|1**

Enables different bitrates per cable. Selections are based on cable type and the user constraint -hstdm_bit_rate. If disabled, a global bitrate that is valid for all cables is used.

**-hstdm_bit_rate** *rate*

For use with HSTDM modules, this argument overrides the default. Changing the bit rate is usually unnecessary. The user can only specify a lower bit rate than the lowest default one. If used in combination with -hstdm_auto_bitrate_selection, the software will not use a bit rate higher than specified value for any cable in the system.

**-hstdm_reset_trace** *traceName*|**none**

For HSTDM module types, assigns the reset signal to the HAPS-70 global reset trace (*traceName*). This signal holds the design in a reset state until training is complete and must be identified prior to performing HSTDM. Generally, the global reset net is the RESETn trace on the HAPS-70 system (a RESETn trace is not supported on HAPS-80 systems). The value none (or NONE) enables HAPS-controlled training through confpro, independent of the reset signal, and is the only supported reset method for HAPS-80 systems.

**-low_latency_mgtdm 1**

Runs MGTDM in performance mode at 20 Gbps.

**-max_ratio** *floatingPoint*

Specifies the highest effective HSTDM/ACPM ratio that can be used by the auto-partitioner. If unspecified, a default ratio based on the selected module type is used; if an invalid ratio is specified, the next available valid ratio is used. When both the -max_ratio and -min_ratio values are the same, only that ratio is used. The -allowed_modules option cannot be used with this option.

**-min_ratio** *floatingPoint*

Specifies the lowest effective HSTDM/ACPM ratio that can be used by the auto-partitioner. If unspecified, a default ratio based on the selected module type is used; if an invalid ratio is specified, the next available valid ratio is used. When both the -min_ratio and -max_ratio values are the same, only that ratio is used. The -allowed_modules option cannot be used with this option.

**-max_mgtdm_ratio** *floatingPoint*

Specifies the highest effective MGTDM ratio that can be used by the auto-partitioner. Highest supported MGTDM ratio is 1024. If unspecified, a default ratio is used based on pre-tdm slack values in MGTDM auto mode; if an invalid ratio is specified, the next available valid ratio is used.

**-min_mgtdm_ratio** *floatingPoint*

Specifies the lowest effective MGTDM ratio that can be used by the auto-partitioner. Lowest supported MGTDM ratio is 64. If unspecified, a default ratio is used based on pre-tdm slack values in MGTDM auto mode; if an invalid ratio is specified, the next available valid ratio is used.

**-qualification_mode** {**all|seq2seq|startseq|endseq**}

Sets the TDM qualification mode as follows:

all – qualifies all nets with the exception of bidirectional clock and asynchronous nets

seq2seq – qualifies only nets that start and end at a sequential element with no intervening paths through an FPGA

startseq – qualifies only nets that start at a sequential element with no intervening paths through an FPGA

endseq – qualifies only nets that end at a sequential element with no intervening paths through an FPGA

**-qualify_clk_enable_paths 0|1**

Qualifies clock enable paths. The default is 0.

**-tdm_fast_default_factor** *factor*

Defines the relationship between the fast and default TDM groups. When the system-route task is in timing mode, nets are automatically assigned to both the fast and default groups based on the timing. factor. For example, when the tdm_fast_default_factor is set to 2.0, the fast-group ratio must be two times smaller (faster) than the default TDM ratio. The system-route task applies this factor.

**-tdm_feedthrough 0|1**
Specifies if qualified feedthroughs can be time domain multiplexed. The default is 0.

**-type** *moduleType*
Specifies the default TDM module type (connection model) to be used for all trace groups. Types include ACPM, HSTDM, HSTDM_ERD, and DIRECT; the default type is HSTDM. Mixing of different TDM types in the same design currently is not supported.

## Examples

```
tdm_control -type hstdm

tdm_control -hstdm_reset_trace mb1.RESETn

tdm_control -hstdm_bit_rate 1000

tdm_control -acpm_fast_clock_trace cpmClk
   -acpm_fast_clock_freq 100

tdm_control -min_ratio 8 -max_ratio 16

tdm_control -allowed_modules {HSTDM_8by2 HSTDM_16by2 16}

tdm_control -qualification_mode seq2seq
```

## Error

An error is reported if the module name or any referenced net does not exist.

# partition_optimization_control

Sets the primary objective of the optimization algorithm and sets the optimization effort level.

## Syntax

**partition_optimization_control**
[**-priority** *objective*]
[**-effort** *level*]
(the next four arguments override specific parts of the -effort option)
[**-max_trials** *number*]
[**-solutions** *number*]
[**-hierarchy_dissolve_pass** *number*]
[**-multi_hop_accuracy 0|1**]
(these are advanced options that are useful for partition exploration)
[**-automatic_replication** *level*]
[**-feedthrough_reduction** *level*]
[**-clock_crossing_reduction** *level*]
[**-illegal_feedthrough_reduction** *level*]
[**-min_utilization_priority** *level*]
[-max_utilization_priority *level*]
[-max_clusters ]
[**-relax_async_constraints 0|1**]
[**-tdm_feedthrough 0|1**]

**-priority** *objective*

Sets the primary objective of the partition optimization algorithm. Accepted values for *objective* are tdm_ratio, nets, multi_hop_path, and slack. The default is nets. The nets option attempts to reduce the total number of nets (including feedthrough nets) that cross FPGA boundaries. The tdm_ratio option attempts to reduce the maximum TDM ratio required for the partition. The multi_hop_path cost attempts to reduce the maximum TDM ratio while considering that nets that are part of multi-hop paths should be routed at the fastest possible TDM ratio.

**-effort** *level*

Sets the effort level of the partition optimization algorithm. Accepted values for *level* are low, normal, and high. The default is normal. The -effort option is equivalent to using several options to set values as shown in the following table.

**-max_trials** *number*
Sets the maximum number of trials for initial solution exploration. The default is set by -effort: -effort low is 150, -effort normal is 300, -effort high is 1000.

**-solutions** *number*
Sets the number of solutions to run in parallel. The default set by -effort: -effort low is 1, -effort normal is 3, -effort high is 6.

**-hierarchy_dissolve_pass** *number*
Sets the number of hierarchy dissolve passes. The default set by -effort: -effort low is 1, -effort normal is 2, -effort high is 2.

**-multi_hop_accuracy 0|1**
Sets the use of either estimation or accurate model for multi-hop paths. Accurate model may use up to 10 times the run time. The default set by -effort: -effort low is 0, -effort normal is 0, -effort high is 1.

**-automatic_replication** *level*
Controls automatic replication of logic during partition optimization. The high option may cause long run times. Does not affect the clock gate replication post-process step. Accepted values for *level* are none, medium, and high. The default is medium.

**-feedthrough_reduction** *level*
Sets the priority of routing feedthrough reduction. Accepted values for *level* are none, low, medium, and high. The default is low. Use this option to encourage the algorithm to reduce feedthroughs at the cost of TDM ratio (or nets).

**-clock_crossing_reduction** *level*
Sets the priority of clock crossing reduction. Accepted values for *level* are none, low, medium, and high. The default is medium. In the early stages of partitioning, when clocks are not completely defined, the clock crossing objective can interfere with getting good results; setting the value to none or low can help. In the later stages, a higher value can help resolve clock crossings during optimization.

**-illegal_feedthrough_reduction** *level*
Sets the priority of illegal_feedthrough (clocks, bidirectional nets, etc.) reduction. Accepted values for *level* are none, low, medium, and high. The default is low. In the early stages of partitioning, when clocks are not completely defined, the illegal feedthrough objective can interfere with getting good results; setting the value to none or low can help. In the later stages, a higher value can help resolve illegal feedthroughs during optimization.

**-min_utilization_priority** *level*

Sets the priority to force the partitioner to honor the minimum utilization constraint. It will trade off the constraint to get better results. Increase the priority to force it to honor the constraint. Accepted values for *level* are none, low, medium, and high. The default is low.

**-max_utilization_priority** *level*

Sets the priority to force the partitioner to honor the maximum utilization constraint. It will trade off the constraint to get better results. Increase the priority to force it to honor the constraint. Accepted values for *level* are none, low, medium, and high. The default is infinity.

**-max_clusters**

Stops optimization if the number of clusters exceeds the limit. The default is low.

**-relax_async_constraints 0|1**

Allows TDM on inter-FPGA asynchronous nets under the same clock group. This option is used along with the report_control -async_nets option. Set the license_feature, FpgaInternalBeta, to enable this option. The default is 0.

**-tdm_feedthrough 0|1**

Enables/disables how feedthroughs affect the partitioning and routing results. If 0, the default, the routing of feedthroughs can only be DIRECT (no TDM). If 1, feedthrough routing can use TDM ratios greater than 1. During the early stages of partitioning, when the inter-FPGA connectivity is not completely defined, you may want to set this argument to 1 to find where the partitioner recommends additional connections.

# pcf_mode

Gets the launch mode of the partitioner and router. Returns partition, route or slr, depending on how the program is launched.

## Syntax

**pcf_mode**

## Example

```
if {[pcf_mode] == "partition"} {

  tdm_control -type HSTDM -min_ratio 8 -hstdm_bit_rate 1400
-qualification_mode all

  clock_gate_replication_control -unlock {FB1.uA FB1.uB …. }

} else {

  tdm_control -type HSTDM -min_ratio 8 -hstdm_bit_rate 1400
-qualification_mode all

}
```

# clock_gate_replication_control

Controls the clock-gate replication algorithm by overriding the bin_attribute -locked file entry to allow clock replication logic to be assigned within a locked bin.

## Syntax

**clock_gate_replication_control** [**-post_process none**|**optimize**] [**-unlock {***binNameList***}**] [**-replicate_hard 0**|**1**]

**-post_process none**|**optimize**
Dissolve replicated clock gates or optimize following replication.

**-unlock {***binNameList***}**
Unlock listed bins to allow clock-gate replication logic to be assigned to a locked bin.

**-replicate_hard 0** | **1**
Replicates cells, including hard assigned cells. The default is **1**.

## Description

Clock gate replication addresses clock skew issues caused by clocks that cross FPGA boundaries. With clock gate replication, clock trees and the requisite gates are replicated in multi-FPGA designs to eliminate the need for manual clock replication. The clock_gate_replication_control command overrides the bin_attribute -locked file entry to allow clock replication logic to be assigned to a locked bin.

Clock gate replication does not automatically replicate MMCMs or PLLs, which must be performed manually using the replicate_cell command.

## Example

```
clock_gate_replication_control -unlock {mb1.uA mb1.uB}
```

## See Also

- -clock_gate_replication argument for *run partition*, on page 144.

# reserve_trace

Removes one or more traces from consideration by the partitioner and router.

## Syntax

**reserve_trace {***traceBusNameList***}**

*traceBusNameList*
A space separated list of the bus traces to be removed from consideration. The list is enclosed in curly braces.

## Example

```
reserve_trace {A[5:0] B[3:0]}
```

## Return Value

Returns the list of trace names. The reserve_trace command issues a warning if any trace name is not found.

# bin_utilization

Enables changing the resource limits of FPGA bins. The command supports bin-group naming. For consistency with other similar commands, you can only specify one resource-value pair per command.

## Syntax

**bin_utilization** [**-bin** *binNameList* |**-all_bins**] [**-all_slr_bins**] [**-all_fpga_bins**] [**-min**]
[**-resource {***resourceName double:value***}**]
[**-resource_ratio {***resourceName***|all** *double:ratio***}**]

**-bin** *binNameList*|**-all_bins**
Specifies the bin or bins to which to apply the change or to apply the change to all bins.

**-all_slr_bins**
Specifies the limits for all SLR bins. Applied only during SLR-partitioning.

**-all_fpga_bins**
Specifies the limits for all FPGA bins Not applied during SLR-partitioning.

**-min**
Specifies a minimum limit for FPGA bin utilization for the partitioner (the partitioner attempts to fill all FPGA bins to at least the specified limit). The -min constraint should only be used with a single resource type which is typically the LUT resource. This constraint is a soft constraint that the partitioner is not required to meet. For example, a constraint value of 0.2 can result in a minimum bin utilization of 0.18.

**-resource {***resourceName double:value***}**
Specifies the value of the limit for the named resource. The valid terms for *resourceName* are LUT, LUTM, DFF, BRAM, DSP, and IO.

**-resource_ratio {***resourceName***|all** *double:ratio***}**
Multiplies the limit for the specified resource by the given ratio. The valid terms for *resourceName* are listed above. Ratio commands are applied to the target system based on the original bin limits. A second ratio specification for the same bin does not further reduce the available resources.

## Example

```
bin_utilization -all_bins -resource_ratio {All 0.8}
# use 80% of the available resources for all bins

bin_utilization -all_bins -resource {LUT 100000 DFF 80000}
# use up to 100,000 of the available LUT resources

bin_utilization -bin U1 -resource_ratio {LUT 0.9}
# use 90% of the available LUT resources

bin_utilization -min -all_bins -resource_ratio {LUT 0.2}
```

## Error

An error is reported:

- when a resource name is invalid
- If a non-existent bin is specified

## Return Value

Returns a list of successfully executed bin names.

# hierarchical_super_bin

Overrides the default super-bin generation.

## Syntax

**hierarchical_super_bin {***binNameList***}** [**-name** *binName*]

*binNameList*
A list of the bins, enclosed in curly brackets, to be included in the super bin (only one super bin can be defined for each command).

**-name** *binName*
The name assigned to the super bin. If any FPGA bins in the system are not defined for a super bin, they are automatically assigned to the OTHER super bin.

## Example

For example, consider an 8-FPGA system with bins mb1.uA, mb1.uB, mb1.uC, mb1.uD, mb2.uA, mb2.uB, mb2.uC, and mb2.uD. Executing the commands

```
hierarchical_super_bin -name SB1 {mb1.uA mb1.uB mb1.uC}
hierarchical_super_bin -name SB2 {mb2.uA mb2.uB mb2.uC}
```

creates three bins: SB1 and SB2 as specified in the commands, and an OTHER super bin containing bins mb1.uD and mb2.uD.

# Board Design and Queries

Constraints in this category support board design and board query; the following constraints are defined:

# design_list_cells

Lists cells on the same hierarchy level within the design according to the specified arguments.

## Syntax

**design_list_cells**
**-cell** *parentCellName* |
**-name {***regexp***}** |
**-assigned 0|1** |
**-unassigned** |
**-type PRIMITIVE|MODULE|ALL** |
**-filter {***resource operation value***}**

**-cell** *parentCellName*
Lists all child cells of the specified parent cell (*parentCellName*).

**-name {***regexp***}**
Filters the list of cells by the specified regular expression. For additional information on regular expressions, see *Regular Expressions, Wildcards, and Special Characters (Standard),* on page 375.

**-assigned 0|1**
Filters the list of cells returned according to if cells are assigned (1) or unassigned (0).

**-unassigned**
Lists all modules with no assigned submodules across multiple hierarchies. Cannot be used with other design_list_cells options.

**-type PRIMITIVE|MODULE|ALL**
Lists cells by type (primitive or module); the argument all (the default) lists both cell types.

**-filter {***resource operation value***}** [**-filter {***resource operation value***} ...**]
Filters the list of cells by the specified resource expression. In the syntax, *resource* is the name of the cell resource (LUT, LUTM, DFF, BRAM, DSP, IO, PORT, or LOGIC), *operation* is one of <, >, ==, >=, <=, or !=, and *value* is the value for the filter expression. The -filter argument can be entered multiple times.

## Description

The design_list_cells command is supported in the PCF editor in both batch and GUI modes and is only available from the system-route state. The command returns a list of instances for a specified parent cell. If a parent cell is not entered, is empty, or a '.' (period), the top-level cell is assumed to be the parent cell. The list of instantiated cells returned is based on the other queries (combinations of queries return the results of the intersection):

- If a -name argument is included, additionally filter by regular expression under the parent cell (or implied root).
- If an -assigned argument is included, additionally filter by property name.
- If a -type argument is included, additionally filter by cell type (PRIMITIVE or MODULE).

## Examples

```
design_list_cells

design_list_cells -type MODULE -assigned 1
   -filter {LOGIC >= 40}

design_list_cells -cell f2 -name {Aor.*}

design_list_cells -unassigned
```

# design_list_nets

Lists nets within the design according to the specified arguments.

## Syntax

**design_list_nets**
**-bidir 0|1** |
**-clock 0|1** |
**-clock_source {***clockSpecificationList***}** |
**-clock_sink {***clockSpecificationList***}** |
**-async 0|1** |
**-interconnect 0|1** |
**-top_io 0|1** |
**-bins {***binList***}** |
**-connections {***binList***}** |
**-name {***regexp***}** |
**-assigned 0|1**
**-buses 0|1**

**-tdm_qualified 0|1**
Filters the list of nets by TDM qualification results.

**-bidir 0|1**
Filters the list of nets by bidir net property.

**-clock 0|1**
Filters the list of nets by clock property.

**-clock _source {***clockSpecificationList***}**
Returns the list of nets which satisfy atleast one of the source clock specifications.

**-clock_sink {***clockSpecificationList***}**
Returns the list of nets which satisfy atleast one of the sink clock specifications.

---

**Note:** -clock_source and -clock_sink options are supported in the System Route state with either, the multi_hop_path mode with option -estimate_timing 1, or the slack mode with option -mapped_timing_-models 1.

---

**-async 0|1**
Filters the list of nets by async property.

**-interconnect 0|1**
Filters the list of nets by interconnect property.

**-top_io 0|1**
Filters the list of nets by top-level I/O property.

**-bins {***binList***}**
Filters the list of nets to show nets that have at least one connection to the specified bins. The -bins argument cannot be used in combination with the -connections argument.

**-connections {***binList***}**
Filters the list of nets to show nets with connections only to the specified bins. The -connections argument cannot be used in combination with the -bins argument.

**-name {***regexp***}**
Filters the list of nets by the specified regular expression. *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.**-assigned 0|1**
Filters the list of nets returned according to if nets are assigned a trace (1) or unassigned (0).

**-buses 0|1**
Filters the list of nets to show nets with connections only to the specified busses.

## Description

The design_list_nets command is supported in the PCF editor in both batch and GUI modes and is available only from the system-route state. The list of nets returned is based on the other queries (combinations of queries return the results of the intersection).

## Examples

```
design_list_nets
```

```
design_list_nets -connections {f3 p1.c1 p1.c2 p2 p3 p4}
```

```
design_list_nets -top_io 1 -tdm_qualified 0 -bidir 0
-clock 0 -async 0 -name {top2.*}
```

The following command sequence uses the net_route command to assign net AorB[0] to board trace f1[0] and then uses the design_list_nets command with the -assigned option to report only the assigned nets:

```
net_route {AorB[0]} -from f3 -to {f2.SL0} -using_trace {f1[0]}

design_list_nets -assigned 1
   expected = "{AorB[0]}";
   execute(cmd, expected);
```

# design_list_ports

Lists ports within the design according to the specified arguments.

## Syntax

**design_list_ports**
**-assigned 0|1** |
**-name {***regexp***}** |
**-bins {***binList***}** |
**-connections {***binList***}**

**-assigned 0|1**
Filters the list of nets returned according to if nets are assigned a trace (1) or unassigned (0).

**-name {***regexp***}**
Filters the list of ports by the specified regular expression. *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.

**-bins {***binList***}**
Filters the list of ports according to connections to the listed bins. The -bins argument cannot be used in combination with the -connections argument.

**-connections {***binList***}**
Filters the list of ports with connections only to the listed bins. The -connections argument cannot be used in combination with the -bins argument.

## Description

The design_list_ports command is supported in the PCF editor in both batch and GUI modes and is available only from the system-route state. The list of nets returned is based on the other queries (combinations of queries return the results of the intersection).

## Examples

```
design_list_ports

design_list_ports -bins {p1.c1 p1.c2 p2 p3 p4}

design_list_ports -assigned 1
```

# design_list_properties

Lists properties within the design according to the specified arguments.

## Syntax

**design_list_properties**
**-cell** *cellName* |
**-net** *netName* |
**-port** *portName*

**-cell** *cellName*
Returns a list of resource-value pairs of area-estimated resources for the specified cell: LOGIC, IO, LUT, LUTM, DFF, BRAM, DSP, MISC, or PORT. ASSIGNED returns 1 if the cell is constrained (by commands such as assign_cell or replicate_cell), and BINS {*binList*} returns a list of the legal bins for the cell. The -cell argument is only available from the partition database state.

**-net** *netName*
Returns a list of the following property-value pairs for the specified net: CLOCK, BIDIR, ASYNC, INTERCONNECT or TOP_IO, ASSIGNED, CELLS {*cellList*}, and BIN {*binList*}. INTERCONNECT indicates that the net is not a port net; TOP_IO indicates a port net. ASSIGNED returns 1 if the net has a net-route or a trace. CELLS {*cellList*} returns a list of the cells connected to the net, and BIN {*binlist*} returns a list of the bins on the net. The -net argument is only available from the system-route database state.

**-port** *portName*
Returns the following property values for the specified port: TO_PORT identifies the bit port toPort() name if present, ASSIGNED is 1 if the bit port is constrained by an assign_port command, and BINS {*binList*} lists the bins for the constrained ports. The -port argument is only available from the system-route database state.

## Description

The design_list_properties command is supported in the PCF editor in both batch and GUI modes.

## Examples

```
design_list_properties -cell {f2.prim_1[0]}
```

```
design_list_properties -net {AorB[6]}
design_list_properties -port {top1[0]}
```

# report_control

The report_control command enables and limits the information printed in the log and report files for the run partition and run system_route commands.

## Syntax

**report_control**
**-async_nets 0|1**
**-direct_nets 0|1**
**-multi_hop_nets 0|1**
**-unconstrained none|ports|cells|all**
**-timing_paths** *integer*
**-unrouted_nets** *integer*
**-key_parameters 0|1**
**-timing_exception_paths** *integer*
**-cell_connectivity** [**0**-**10**]
**-tdm_metrics 0|1**
**-subsystem_keepbufs 0|1**
**-adv_hop_report 0|1**

**-async_nets 0|1**
Reports which asynchronous nets are qualified for TDM and displayed in the System Route log. This option is used along with the partition_optimization_control -relax_async_constraint option. Set the license_feature, FpgaInternalBeta, to enable this option. The default is 0.

**-direct_nets 0|1**
Reports the nets from the Partition Results and Global Routing log sections that are routed without TDM. The default is 0.

**-multi_hop_nets 0|1**
Reports all nets from the Partition Results log section that are part of a multi-hop path. The default is 0.

**-unconstrained none|ports|cells|all**
Reports all cells, ports, or all cells and ports (all) from the Constraint Summary log section that do not include user-defined constraints. The default is none.

**-timing_paths** [*integer*]
Limits the number of paths reported in the timing_partition.rpt or timing_route.rpt file to *integer*. The default is 10 which reports the first 10 paths; specifying a value of 1023 or greater causes all paths to be reported.

**-unrouted_nets** [*integer*]
Limits the number of unrouted nets reported in the Global Routing log section to *integer*. The default is 100 which reports the first 100 unrouted nets; specifying a value of 1023 or greater causes all unrouted nets to be reported.

**-key_parameters 0|1**
Reports the key parameters for the run in comma separated value (csv) format. The default is 1 (report key parameters).

**-timing_exception_paths** *integer*
Specifies the maximal number of paths reported per timing exception. Output is written to report file timing_exceptions.rpt. If the value is 0, no timing exception report is generated.

**-cell_connectivity** [**0**-**10**]
Enables the listing of cells in the partition report, in the cell_connections section. The default value, **0**, disables this argument. Values 2 to 10 provides the number of top connections to cells.

**-tdm_metrics 0|1**
Writes the contents of the tdm_nonqualified.rpt to the metrics database during SRP build. Default is disabled.

**-subsystem_keepbufs 0|1**

Reports keepbufs added to preserve subsystem connectivity and their assignments. The default is 0.

**-adv_hop_report 0|1**

Writes advanced multi-hop report. The default is 0.

## Examples

```
report control -direct_nets 1 -unrouted_nets 50
```

# reset_synchronize

Controls synchronization of global reset or other asynchronous signals.

## Syntax

**reset_synchronize**
[**-flop** *resetSyncFlop*|**-net** *resetSyncNet* **-clock** *syncClock*]
[**-extra_pipeline_stages** *integer*] [**-init 0**|**1**] [**-force_repl** |**-repl_bins {***binList***}**]

**-flop** *resetSyncFlop*
Identifies the design flip-flop for synchronization across all FPGAs.

**-net** *resetSyncNet* **-clock** *syncClock*
Identifies the design net for synchronization across all FPGAs; the -clock argument identifies the clock associated with the net and is required. The clock net specified must be available at the top level hierarchy. You can specify design ports and internal design nets as reset nets. The net must directly connect to the flip-flop reset loads; you cannot use this command for nets that connect to combinatorial loads, such as IBUFG or logic inverters. For such nets, make the net directly connected to the flip-flop reset ports the *resetSyncNet*.

**-extra_pipeline_stages** *integer*
Specifies the number of additional pipeline flip-flops to be added to each FPGA. The default is 0.

**-init 0|1**
Specifies the initial value of the flip-flops (0 or 1) for the reset propagation chain. 0 sets active low reset. 1 sets active high reset.

**-force_repl**
Replicates the synchronize *resetSyncFlop* to all non-locked FPGA bins, regardless of the loads on the reset net.

**-repl_bins {***binList***}**
Replicates the synchronize flip-flop into the specified list of FPGA bins.

## Description

The reset_synchronize command provides a way to synchronize global resets or other asynchronous signals in HAPS-80 systems. It supports both active-low (-init 0) and active-high (-init 1) resets.

Specify a unique net for each clock domain. For multiple clock domains and unique reset synchronization per clock domain, a unique net for each clock domain must be created with syn_keep (syn_keep, on page 667 in the *HAPS ProtoCompiler Compiler and Mapper Guide* ), inverters, or buffers.

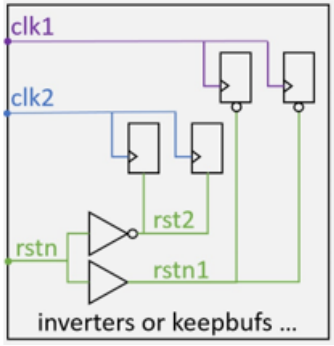
For more information on reset synchronization, see Synchronizing Resets, on page 280 in the *User Guide*.

## Example:

```
reset_synchronize -flop sync2 -extra_pipeline_stages 2
   -init 1 -repl_bins (mb1.uA mb1.uB mb1.uC mb1.uD}
```

Top-level reset and clock nets in multiple clock regions:

```
reset_synchronize -net {rstn1} -clock {clk1} -init {0}
reset_synchronize -net {rst2} -clock {clk2} -init {1}
```




Design reset and clock FPGA ports with active-low reset:

```
reset_synchronize -net {rstn} -clock {clk} -init {0}
```

Design synchronous reset and clock nets with active-low reset

```
reset_synchronize -net {rstn1} -clock {clk1} -init {0}
reset_synchronize -net {rstn2} -clock {clk2} -init {0}
```

Design reset and MMCM clock nets with active-high reset

```
reset_synchronize -net {rst1} -clock {mmcm_clk1} -init {1}
reset_synchronize -net {rst2} -clock {mmcm_clk2} -init {1}
```

# timing_control

Controls timing graph generation and constraint application.

## Syntax

**timing_control**
[**-auto_constrain_ports** *value*]
[**-port_delay** *delayLength*]
[**-zero_delay** *delayType*]

**-auto_constrain_ports** *value*]
Automatically constrain top-level ports. *Value* may be one, inputs, output, and all. The default is all.

**-port_delay** *delayLength*
Specifies the delay value (in nsec) for automatic port constraints. The default is 10.0.

**-zero_delay** *delayType*
Controls the delays for intra-FPGA connections. *delayType* may be 0 or 1. If 0, the default, use estimated delays. If 1, use zero for all intra-FPGA delays.

# tss_list_bins

A query mechanism for bins.

## Syntax

**tss_list_bins**
**-type** [*binType*] |
**-trace** *traceName* |
**-name {***regexp***}** |
**-locked 0|1**

**-type** [*binType*]
Returns a list of bins of the specified type. Acceptable type values for *binType* are FPGA, EXTERNAL, PORT, and ALL (the default).

**-trace** *traceName*
Returns a list of bins associated with the named trace (*traceName*).

**-name {***regexp***}**
Returns a list of bins based on the regular expression query. See *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.

**-locked 0|1**
Returns a list of bins according to their locked status. Setting the -locked argument to 1 lists the locked bins; the default for -locked is 0.

## Description

The tss_list_bins command returns a list of bins in the TSS. The command is supported in the PCF editor in both batch and GUI modes and is available only from the partition state. The list of bins returned is based on the other queries, and combinations of queries return the results of the intersection:

- Without any arguments, tss_list_bins returns all bins
- If -type is specified, filters by bin type
- If -trace is specified, filters by connected traces
- If -name is specified, filters by regular expression

## Examples

```
tss_list_bins -locked 1
```

```
tss_list_bins -type EXTERNAL
```

# tss_list_functions

A query mechanism for functions.

## Syntax

**tss_list_functions -name {***regexp***}**

**-name {***regexp***}**
Returns a list of functions based on the regular expression query. See *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.

## Description

The tss_list_functions command returns a list of functions in the TSS based on a regular expression. The command is supported in the PCF editor in both batch and GUI modes and is available only from the partition state.

## Examples

```
tss_list_functions
```

```
tss_list_functions -name {GCLK[02]}
```

# tss_list_pins

A query mechanism for pins.

## Syntax

**tss_list_pins**
**-bin** *binName* |
**-trace** *traceName* |
**-name {***regexp***}**

**-bin** *binName*
Returns a list of pins associated with the specified bin.

**-trace** *traceName*
Returns a list of pins connected to the specified trace.

**-name {***regexp***}**
Returns a list of pins according to the regular expression query. See *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.

## Description

The tss_list_pins command returns a list of pins in the TSS. The command is supported in the PCF editor in both batch and GUI modes and is available only from the partition state. The pin name is hierarchal and uses the following syntax:

*binName/connectorName.pinName*

The list of pins returned is based on the other queries, and combinations of queries return the results of the intersection:

- Without any arguments, tss_list_pins returns all pins
- If -bin is specified, filters by pins from that bin
- If -trace is specified, filters by connected pins
- If -name is specified, filters by regular expression

## Examples

```
tss_list_pins
```

```
tss_list_pins -bin {f1.SLR1} -trace {f4[7]} -name {J8}
```

# tss_list_properties

Returns a list of property value pairs for a given trace, bin, or pin.

## Syntax

**tss_list_properties**
**-name** [*propertyName*] |
**-trace** *traceName* |
**-bin** *binName* |
**-pin** *pinName* |
**-trace_group** *traceGroupName*

**-name** [*propertyName*]
Returns a list of properties in the TSS. If the -name argument is specified, returns the value of the requested property (*propertyName*).

**-trace** *traceName*
Returns a list of property-value pairs for the specified trace. Trace properties are: TRACE_GROUP, CONNECTIVITY, CONNECTION_MODEL, and SOURCE_BIN.

**-bin** *binName*
Returns a list of property-value pairs for the specified bin. Bin properties include the modified limit resources (DFF, BRAM, DSP, IO, LUT, and LUTM) and other properties such as TYPE, LOCKED, and USED_PINS.

**-pin** *pinName*
Returns a list of property-value pairs for the specified pin. Pin properties are: SLR, CLOCK_CAPABLE, and DIFFERENTIAL.

**-trace_group** *traceGroupName*
Returns a list of property-value pairs for the specified trace group.

## Description

The tss_list_properties command returns a list of properties in the TSS. The command is supported in the PCF editor in both batch and GUI modes and is available only from the partition state.

The list of pins returned is based on the other queries, and combinations of queries return the results of the intersection:

- Without any arguments, tss_list_pins returns all pins

- If -bin is specified, filters by pins from that bin
- If -trace is specified, filters by connected pins
- If -name is specified, filters by regular expression

## Examples

```
tss_list_pins
tss_list_pins -trace {f4[7]}
tss_list_pins -bin {f1.SLR1} -trace {f4[7]} -name {J9.*}"
```

# tss_list_trace_groups

A query mechanism for trace groups.

## Syntax

**tss_list_trace_groups** [**-name {***regexp***}**]

**-name {***regexp***}**
Returns a list of pins according to the regular expression query. See *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.

## Description

The tss_list_trace_groups command returns a list of trace groups in the TSS based on a regular expression. The command is supported in the PCF editor in both batch and GUI modes and is available only from the partition state.

## Examples

```
tss_list_trace_groups -name {f[247]|LK}
```

# tss_list_traces

A query mechanism for traces.

## Syntax

**tss_list_traces**
**-trace_group** *traceGroupName* |
**-function** *functionName* |
**-bins {***binList***}** |
**-connections {***binList***}** |
**-name {***regexp***}**

**-trace_group** *traceGroupName*
Returns a list of traces within the specified trace group.

**-function** *functionName*
Returns a list of traces with the specified function name.

**-bins {***binList***}**
Returns a list of traces that are connected to the specified bins. The -bins argument cannot be used in combination with the -connections argument.

**-connections {***binList***}**
Returns a list of traces that are only connected to the specified bins. The -connections argument cannot be used in combination with the -bins argument

**-name {***regexp***}**
Returns a list of pins according to the regular expression query. See *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375.

## Description

The tss_list_traces command returns a list of traces in the TSS. The command is supported in the PCF editor in both batch and GUI modes and is available only from the partition state.

The list of pins returned is based on the other queries, and combinations of queries return the results of the intersection:

- Without any arguments, tss_list_traces returns all traces
- If -trace_group is specified, filters the list of traces by trace group name.

- If -bin is specified, filters the list of traces by the list of bins.
- If -trace is specified, filters by connected pins
- If -name is specified, filters by regular expression

## Examples

```
tss_list_traces -bins {f1.SLR1}
tss_list_traces -connections {f1.SLR1 f3}
tss_list_traces -trace_group f5 -connections {f1.SLR1 f3}
```

# zcei_assign_route

Enables use of Zebu transactors on HAPS-80.

## Syntax

**zcei_assign_route**
**-master_bin** *fpga*
**-clock_source** *name*
**-trace** *value*
**-gclk** *gclk*

**-master_bin** *fpga*
Specifies the FPGA bin to which the CCM master will be assigned.

**-clock_source** *list*
Specifies a list of ZCEI clock sources to be used by the -trace and -gclk arguments. This argument can be omitted if the clock list has a size of 1. The value *list* for -clock_source and -gclk must be the same size.

**-trace** *value*
Specifies ZCEI control nets to be assigned to global clock. Currently only cclock_base is valid value for this option, which is also the default.

**-gclk** *list*
Specifies a list of global clock traces to be used for ZCEI clock nets. The value *list* for -clock_source and -gclk must be the same size.

## Description

The zcei_assign_route command is used to specify which global clock traces to use for ZCEI clock control module signals.

## Example

# Auto Hierarchy Dissolve Controls

The partitioner follows a two-step process when it dissolves cells:

- An initial dissolve step attempts to ensure that design partitioning is feasible, by dissolving cells that are too big to fit in one FPGA or cells that are connected to a large number of external ports.
- A hierarchy-pass dissolve step that dissolves cells that are likely to improve the partitioning results.

## Hierarchy Dissolve Controls

The dissolve_control command allows the upper and lower bounds to be defined when dissolving hierarchical modules. You can also set limits to prevent modules with a large amount of random logic from being dissolved. Random logic is any logic that is not a hierarchical module. To illustrate, consider a module that contains 10,000 LUTs. If the module contains 100 LUTs of random logic, and the rest of the logic is in other hierarchy modules, there is only a small partitioning penalty. However, if all 10,000 LUTs are directly instantiated within the module, it might be better to exclude these cells which could slow down the partitioner and adversely affect the results.

User-specified dissolve constraints (cell_attribute -dissolve 1 and cell_attribute -dissolve 0) always override the automatic controls. Some commands (such as assign_cell) create dissolve constraints as a side effect.

## Dissolve Constraint Precedence

The precedence of hierarchy constraints is shown below. The partitioner can dissolve other modules not specifically covered in the list below, as needed.

1. cell_attribute -dissolve 0 ➔ never dissolve
2. cell_attribute -dissolve 1 ➔ dissolve immediately
3. module greater than *maxValue* ➔ dissolve immediately
4. local logic/total logic greater than local_logic_ratio ➔ never dissolve
5. local logic greater than random logic limit ➔ never dissolve
6. module less than *minValue* ➔ never dissolve

# dissolve_control

Defines the upper and lower bounds for dissolving hierarchical modules. The attribute is ignored during run system_route. See *Auto Hierarchy Dissolve Controls*, on page 208 for details about how the partitioner dissolves hierarchy.

## Syntax

**dissolve_control** [**-auto 0**|**1**]

**-auto 0|1**
When set to 1, the partitioner ignores all other dissolve control options. The default value is 1. When set to 0, dissolve_control options are honored.

## Example

```
dissolve_control -auto 1
```

## Return Value

Returns 1.

# Cell and Port Attributes and Assignments

- Constraints in this category support hierarchical bin naming and control the clustering of logic and port cells; the following constraints are defined:assign_cell, on page 211
- replicate_cell, on page 214
- cell_attribute, on page 216
- cell_utilization, on page 217
- cluster, on page 219
- port_attribute, on page 221
- assign_port, on page 222
- cluster_port, on page 226

# assign_cell

Specifies the set of legal FPGA bins for a set of cell instances. Each specified cell can occupy any of the specified bins. The instance is restricted from clustering with other cells and can never be dissolved; the parent modules are recursively dissolved. The command is not applicable during run system_route.

## Syntax

**assign_cell**
*nameList*
*binGroupNameList*
**-soft**
**-hard**
**-net**
**-clock_pair**

*nameList*
Specifies a list of cell instances to assign; if -net is also specified, *nameList* is a list of nets. When more than one net or cell is specified, the names must be enclosed in curly braces and separated by spaces.

*binGroupNameList*
The name of the target FPGA bin. If more than one bin is specified, the bin names must be enclosed in curly braces and separated by spaces. A bin name cannot map to a port bin and, if the bin is locked, only one target bin is allowed. If the bin is a black-box bin, only objects from the black box netlist can be assigned.

**-soft**
Including the optional -soft argument allows this constraint to be removed during partitioning for improved results. If both -soft and -hard are set, -hard takes precedence.

**-hard**
If this argument is specified, the partitioner may not change this assignment. If both -soft and -hard are set, -hard takes precedence.

**-net**
Interprets *nameList* as a list of net names; the partitioner assigns all cells connected to the listed nets to the specified bins.

**-clock_pair**

Interprets *nameList* as a list of clock pairs. The partitioner assigns all cells that are part of the clock pair to the specified bins. Clock names are specified with the create_clock or create_generated clock commands. Use :r or :f to specify rising or falling edge. Wild cards are supported.

## Example

```
assign_cell myCell mb1.uA

assign_cell q_nic4 {mb1.uA mb1.uB}

assign_cell {cell_a1 cell_b2} mb2.uC

assign_cell {accum_inst.data[7:0]} mb1.uA
```

## Error

An error is reported:

- if any assignment exceeds the capacity of all specified bins
- if this constraint conflicts with any previous constraint
- if the named cell cannot be found

## Return Value

Returns a list of successfully assigned cells.

# replicate_cell

Forces replication of each specified cell into each of the specified bins. The command is particularly useful for replicating clock buffers into each of the FPGAs where they are used. If, after partitioning, the replicated cell has no output connections, it is removed. Cells are restricted from clustering with other cells and are never dissolved; parent modules are recursively dissolved. The command is not applicable during run system_route.

## Syntax

**replicate_cell {***cellNameList***}** [**-all_bins** ]|**{***binGroupNameList***}** [**-constrain_outputs**] [**-force**] [**-soft**] [**-hard**]

**{***cellNameList***}**
The name of one or more cells to be replicated. If more than one cell is specified, the cell names must be enclosed in curly braces and separated by spaces.

**-all_bins** | **{***binGroupNameList***}**
Replicates cells in all bins or specifies a list of target bin groups for replication. Bins must be FPGA bins and a minimum of two bins must be specified (or -all_bins). The bin list entries are enclosed in curly braces and separated by spaces.

**-constrain_outputs**
Restricts assignment of each cell driven by the outputs of the replicated cells to the same set of bins.

**-force**
Forces cell replication even when the cell does not include output connections in the target bin.

**-soft**
Including the optional -soft argument allows this constraint to be removed during partitioning for improved results.

**-hard**
If specified the partitioner may not change this assignment (default).

## Example

```
replicate_cell  myCell {FPGA1 FPGA2}

replicate_cell -constrain_outputs{cell1 cell2}{FPGA1 FPGA2}
```

```
replicate_cell {cell1 cell2} -all_bins
```

## Error

An error is reported:

- if it conflicts with any previous constraint
- if a named cell cannot be found

## Return Value

Returns a list of successfully assigned cells.

# cell_attribute

Controls the dissolving and replication of specific cells. The command is ignored during run system_route.

## Syntax

**cell_attribute {***cellNameList***}** [**-dissolve 0|1|auto**] [**-auto_replicate 0|1**]

**{***cellNameList***}**
A list of the cells to be dissolved or replicated. The list must be enclosed in curly braces and the individual entries in the list must be separated by one or more spaces.

**-dissolve 0|1|auto**
When set to 1, dissolves the specified cells and their parents immediately after reading the netlist (-dissolve is implied for the parents of cells constrained by any assignment specification). When set to 0, prevents the named cells from being dissolved and is implied for cells constrained by any assignment specification. The auto argument (the default) allows an algorithm to determine the dissolving of cells.

**-auto_replicate 0|1**
When set to 1, enables auto replication of the listed cells.

## Example

```
cell_attribute {mycell1 mycell2} -dissolve 1
```

## Error

An error is reported:

- if it conflicts with any previous constraint
- if any of the cells cannot be found

## Return Value

Returns a list of successful cell names.

# cell_utilization

Overrides the usage estimates for a particular hierarchy cell or for the top-level module.

## Syntax

**cell_utilization** {**-top** |**-cell** *cellName*}
[**-resource {***resourceName value***}** [**... {***resourceName value***}**]] |
[**-bin** *binName* [**-resource_ratio {***resourceName double:value***}**]]

**-top**|**-cell** *cellName*
Specifies the cell or top-level module to be overridden.

**-resource {***resourceName value***}**
Specifies an explicit value for a resource. The complete argument can be repeated. The valid terms for *resourceName* are LUT, LUTM, DFF, BRAM, DSP, and IO.

**-bin** *binName* [**-resource_ratio {***resourceName double:value***}**]
Specifies the bin and a usage ratio for that bin. The complete -resource_ratio argument can be repeated. The valid terms for *resourceName* are listed above.

## Examples

```
cell_utilization -top -resource {LUT 10.0e6} {DFF 30.0e6}

cell_utilization -cell top.next.A -resource {BRAM 55}

cell_utilization -cell top.next.A -bin mb1.uA
   -resource_ratio {BRAM 0.55}
```

## Return Value

The cell_utilization command returns 1 if successful, 0 if unsuccessful.

## Error

An error is reported:

- if *cellName* is not found
- if *resourceName* is invalid

- if *binName* is not found

# cluster

Clusters cells together. While cells are clustered, they always are assigned to the same bin. This attribute is ignored during run system_route.

## Syntax

**cluster {***cellNameList***}** [**-soft**] [**-net** *nameList*] [**-clock_pair** *nameList*] [**-clock_pair_max_distance** *value*]

*cellNameList*
A list of cell names to be clustered. The list must be enclosed in curly braces and the individual entries in the list must be separated by one or more spaces

**-soft**
Including the -soft attribute allows this constraint to be removed during partitioning for improved results.

**-net**
If specified, interprets *nameList* as a list of net names. The partitioner will cluster together all cells connected to those nets.

**-clock_pair**
If specified, interprets *nameList* as a source-sink pair of clock names. The partitioner will cluster all cells that are part of that clock pair. Clock names are those specified in create_clock or create_generated clock commands. Use :r or :f to specify rising or falling edge. Wild cards are supported. Examples:
{clkin:r clkin:f}
{clk*:r clk*:f}
{clkin:* clkin:*}
{clkin clkin}
The last two examples are different ways of doing the same thing.

**-clock_pair_max_distance**
Implicitly clusters based on the current dissolve state each individual clock pair with path constraints less than the specified threshold. *value* is time in nanoseconds.

## Example

```
cluster {cell1 cell2} -soft
```

## Error

An error is reported:

- if it conflicts with any previous constraint
- if any of the cells cannot be found

## Return Value

Returns a list of successful cell names.

# port_attribute

Adds an attribute to a group of top-level ports.

## Syntax

**port_attribute** [**-ignore**] {*portBusNameList*}|**-port** *type*

**-ignore**
Causes the partitioner and router to ignore the specified port or ports and any connected nets.

{*portBusNameList*}
Lists the top-level ports. If more than one top-level port is specified, the port names must be enclosed in curly braces and separated by spaces.

**-port** *type*
Applies the attribute to all ports of the specified type. Valid types are list (default), input, output, bidir, and all. Bidir ports are not included in the input and output port sets.

## Return value

Returns a value of 1 when successful.

## Example

```
port_attribute -ports input;
   #apply attribute to all input ports

port_attribute -ignore -ports input;
   #ignore input ports

port_attribute -ignore -ports bidir;
   #ignore bidir ports

port_attribute {a b c[3:0]}
   #apply attribute to ports listed (a, b, and c[3:0])
```

# assign_port

Specifies the set of legal port bins for a top-level port, and allows exact pins to be defined on a top-level port.

## Syntax

**assign_port {***portBusNameList***}** [**{***binGroupNameList***}**] [**-soft**] [**-hard**] [**-constrain_cells 0**|**1**] [**-trace {***traceNameList***}** |**-pin {***binPinNameList***}**]

**{***portBusNameList***}**
The name of one or more ports. If more than one port is specified, the port names must be enclosed in curly braces and separated by spaces. The command supports hierarchical pin naming. The number of pin names specified (explicitly or through bus notation) must match the number of assigned ports. Each specified port is assigned to its corresponding pin.

**{***binGroupNameList***}**
Specifies the set of legal connector port bins. Each named bin must be a port bin or an external bin. If more than one port bin is specified, the port connector names must be enclosed in curly braces and separated by spaces. For the run system_route phase, you cannot specify multiple bins.

**-soft**
Allows the partitioner to remove the constraint when it improves results. Without this argument, the default is -hard, and the partitioner treats assignments as hard definitions.

**-hard**
(Default). Treats all assignments as hard definitions. To improve results by allowing the tool to choose whether to honor the constraint, specify -soft.

**-constrain_cells 0 | 1**
Constrains cells that are connected to the specified object to FPGA bins that are connected to the specified *traceNameList* trace. The default is **0** (when assigning to portbin*Name*/*pinName*, the default is **1**).

**-trace {***traceNameList***}** |**-pin {***binPinNameList***}**
A list of port trace names or port pin names.

The assignments.pcf file generated by the partitioner might contain additional options to the assign_port statement, like -fwd. These options are not supported user options.

## Example

```
assign_port IN[25:12] {Conn1 Conn2}

assign_port {IN[25:12] CLK} {Conn1 Conn2}

assign_port IN[11:0] Conn1/J.A[11:0]
   # assign to specific pins

assign_port {IN[11:0]} -trace {T[11:0]}

assign_port {myport1} {portbin_c_22}
   # assign to any pin of portbin_c_22
assign_port {myport2} {portbin_c_22/A[5]}
   # assign to pin A[5] of portbin_c_22
```

For HAPS-100 the TSS user-specified clock name should be specified in the PCF. This is different from fixed GCLK names for HAPS-80.

```
assign_port clk –trace myclk1
```

## Error

An error is reported:

- if the assignment exceeds the capacity of the bin
- if it conflicts with any previous constraint
- if the named port cannot be found

## Return Value

Returns a list of successfully assigned ports.

# assign_virtual_port

Assigns a design port to a virtual port controlled through Confpro.

## Syntax

**assign_virtual_port**
**-port**
**-type**
**-bin**

**-port**
Specifies a design port.

**-type**
Specifies a supported virtual port tag.

**-bin**
Specifies an FPGA bin for assignment.

# auto_tss_control

Controls automatic cabling.

## Syntax

**auto_tss_control**
**-enable 0|1**

**-enable 0|1**
The default, 0, disables automatic cabling.

When automatic cabling is enabled, a new TSS file with optimal connections is generated. See Defining Connections with Auto-Cabling, on page 197 in the *User Guide* for details.

# cluster_port

Clusters top-level ports together. When clustered, top-level ports are always assigned to the same port bin. This attribute is ignored during run system_route.

## Syntax

**cluster_port {***portBusNameList***}** [**-soft**]

**{***portBusNameList***}**
A list of port bus names to be clustered. The list must be enclosed in curly braces and the individual entries in the list must be separated by one or more spaces.

**-soft**
Uses the cluster definition during the initial solution, but allows the definition to be subsequently dissolved by the partitioning engine (in the absence of this option, the cluster definition is respected throughout the partitioning process).

## Example

```
cluster_port {p1 p2} -soft
```

## Error

An error is reported:

- if it conflicts with any previous constraint
- if any of the ports cannot be found

## Return Value

Returns a list of successful port names.

# Net Attributes, Grouping, and Assignments

The partitioner is cell focused and for most constraints, using cell and port assignments is recommended. It is, however, not always possible to get the required results using only cell and port assignments.

The commands in this section support net groups and net assignments and include the following:


- cluster_net, on page 228
- net_attribute, on page 229
- assign_global_net, on page 233
- trace_group_attribute, on page 235
- net_route, on page 237

# cluster_net

Assigns a set of nets to a named cluster which can be referenced by name in other net commands. In the partitioner, the drivers of the specified nets are clustered to ensure that the nets remain together. The router always attempts to assign nets to the same cluster group.

## Syntax

**cluster_net** *clusterName* **{***netNameList***}** [**-hard**]

*clusterName*
The name of the cluster group.

**{***netNameList***}**
A list of the net names to be included in the cluster group. The net names must be enclosed in curly braces and separated by spaces.

**-hard**
If the -hard attribute is specified, the partitioner attempts to keep the nets together by also clustering the drivers of the nets. The router attempts to assign the nets of a hard net group to the same channel, but may break up the group if the constraint is not feasible.

## Example

```
cluster_net MyCluster {a[5:0] top.b}
   # creates a cluster group with seven elements
```

## Error

An error is reported in these cases:

- If the net is not found
- If a net name has been previously used

## Return Value

Returns the list of net names assigned to the cluster.

# net_attribute

Changes the attributes for a set of nets to control post-partitioned routing.

## Syntax

**net_attribute {***netNameList***}** [**-function** *functionName*]
[**-differential** *netNegName*] [**-diffsingle 0**|**1**] [**-is_clock 0**|**1**|**auto**]]
[**-is_bidir 0**|**1**|**auto**] [**-is_async 0**|**1**|**auto**] [**-tdm_qualified 0**|**1**]
[**-feedthrough_allowed 0**|**1**] [**-routing_priority** *integer*]
[**-tdm_group [TDM|DIRECT|<***tdmModuleName***>]|{<sinkBin>**
**[TDM|DIRECT|<***tdmModuleName***>]}]**
[**-replicate 0**|**1**] [**-ignore**]

**{***netNameList***}**
A list of net names.

**-function** *functionName*
Assigns the list of nets to the named function to restrict the trace assignment to traces with the same function. The function must have been previously defined to a trace group.

**-differential** *netNegName*
Includes only a single net in *netNameList*. This net is marked as the P net of a differential pair with the *netNegName* net marked as the N net of the pair. All other options (-function, -connection_model, etc.) are applied to both the P and N nets. (Not supported for HAPS-100, see Examples)

**-diffsingle 0|1**
Routes nets on the P trace and preserves the N trace. The prototyping tool adds a differential buffer, connects the N net to the buffer, and uses the N trace as the pin location (currently applies only to GCLK clock nets). The default is 1 (preserves the N trace).(Not supported for HAPS-100).

**-is_clock 0|1|auto**
Specifies whether the net is treated as a clock signal (1), or not (0), overriding what the tool automatically inferred (auto). Treating it as a clock signal includes minimizing clock crossings and illegal feedthroughs, and routing on traces with clock-capable terminal FPGA pins. The default is auto.

**-is_enable 0|1|auto**
Specifies whether to override the automatic enable detection. The default is auto.

**-is_bidir 0|1|auto**
Specifies whether the net is treated as a bidirectional signal or tristate (1), or not (0), overriding what the tool automatically inferred (auto). Treating it as a bidirectional signal includes minimizing illegal feedthroughs and avoiding net splitting and routing without TDM. The default is auto.

**-is_async 0|1|auto**
Specifies whether the net is treated as an asynchronous reset (1), or not (0), overriding what the tool automatically inferred (auto). Treating it as an asynchronous reset includes minimizing illegal feedthroughs and routing without TDM. The default is auto.

**-tdm_qualified 0|1**
Indicates if the listed nets are qualified for time-domain multiplexing.

**-feedthrough_allowed 0|1**
Indicates if the listed nets are allowed to pass through intermediate devices.

**-replicate 0|1**
Enables/disables automatic clock replication for cells connected to this net; default is enable replication (1).

**-routing_priority** *integer*
Sets the routing priority for the listed nets. The integer value range is from 1 to 255 (highest priority). Higher the value, higher the priority. Default is 0 and indicates the nets that have not been assigned any priority.

**-tdm_group [TDM|DIRECT|<***tdmModuleName***>]|{<sinkBin> [TDM|DIRECT|<***tdmModuleName***>]}**
Assigns nets to the specified group.

With TDM, all nets in the group are assigned the same ratio per cable, based on available traces. When timing-driven routing optimization is enabled for system route (-estimate_timing), the nets in the TDM group are assigned different ratios based on the slack and available traces. If no nets have a defined net_attribute, system route attempts to assign the lowest TDM ratio based on the number of available nets versus the number of available traces.

The DIRECT argument specifies no pin muxing.

The *tdmModuleName* argument assigns the nets to the named TDM module. The optional *sinkBin* argument assigns the net to a specific TDM group for the specified sink bin only.

The *tdmModuleName* argument with ratio is used for HSTDM and MGTDM. The format varies based on the usage.

| | |
|---|---|
| HSTDM (Default - Depending on the connection, this can be single-ended for differential) | HSTDM_[8,16,24…112,120,128,160,192,224,256] |
| HSTDM ERD (HSTDM with Error Detection) | HSTDM_ERD_[7,15,22…105,112,120,150,180,210,240] |
| HAPS-80_MGTDM (Manual insertion only) | HSTDM_MGT_[64,128,256,512,1024] |
| HAPS-100 MGTDM (Automatic and manual insertion) | HSTDM_MGT_[64,128,256,512,1024] |

**-ignore**
Ignores the selected nets.

## Examples

```
net_attribute {clk1  clk2} -function CLOCK

net_attribute {u4.dat1[39:0] u4.dat2[39:0] u4.dat3[39:0]}
   -function MGT -tdm_group HSTDM_MGT_128
```

HAPS-100 has the following PCF command differences from HAPS-80.

For single-ended clock syntax **-diffsingle** is not supported for HAPS-100.

```
//HAPS-100
net_attribute clk -function GCLK

//HAPS-80
net_attribute clk -function GCLK -diffsingle
```

For differential clock syntax, **-differential** is not supported for HAPS-100 and the ignore command must be added for N port.

```
//HAPS-100
net_attribute clk_P -function GCLK
assign_global_net clk_P myclk1
port_attribute -ignore clk_N
```

```
//HAPS-80
net_attribute clk_P -differential clk_N -function GCLK
assign_global_net clk_P fb1.GCLK1
```

## Error

An error is reported if the net or function is not found.

## Return Value

Returns the list of nets to which attributes were applied.

# assign_global_net

Assigns a global clock or asynchronous reset net to an FPGA through the low-skew GCLK1 to GCLK12 traces of the HAPS system.

If the driver of the net is inside the design, clock signals are routed out of the FPGA through the GCLK MUX and then back into the FPGA and to the other FPGAs on the GCLK trace.

If the driver of the net is external to the design, it is a top-level port. The source of the trace must be listed in the TSS file (for example, PLL).

Any local loads for a signal are driven from the external GCLK trace. GCLK0 is reserved for the system 100 MHz clock.

## Syntax

**assign_global_net** *netName gclkName* [**-constrain_cells 0**|**1**]

*netName*
The name of the net to be assigned.

*gclkName*
The name of the GCLK trace.

**-constrain_cells 0** | **1**
Constrains cells that are connected to the specified *netName* to FPGA bins that are connected to the specified *gclkName*. The default is **0** (when assigning to portbin*Name*/*pinName*, the default is **1**).

## Example

```
assign_global_net my_clk GCLK0
```

Note for HAPS-100 that TSS user-specified clock name should be specified in the PCF. This is different from fixed GCLK names for HAPS-80.

```
assign_global_net clk myclk1
```

## Error

An error is returned:

- if the net or global trace cannot be found.

- if the net or global trace is already assigned.
- if GCLK trace source (FPGA vs. PLL) doesn't match the net source (design vs toplevel port).

## Return Value

Returns the net name if successful, else an empty list.

# trace_group_attribute

Changes the attributes for a group of traces to control post-partition routing.

## Syntax

**trace_group_attribute {***traceGroupName***}** [**-connection_model** DIRECT] [**-function** *signalFunctionName* [**-hard 1**|**0**]] [**-from** *binName*]

**{***traceGroupName***}**
Specifies the name of the trace group to change. You can find the names of trace groups in the tss log report.

**-connection_model** ACPM | HSTDM | HSTDM_ERD | DIRECT
Specifies the default connection model (TDM module type) for the trace group: ACPM, HSTDM, HSTDM_ERD, or DIRECT. If there are not enough traces between FPGAs, the tool automatically sets the TDM connection model based on the kind of bank. Use the DIRECT keyword to manually specify direct connections (no TDM) between FPGAs.

**-function** *signalFunctionName* [**-hard 1**|**0**]
Annotates the traces with *signalFunctionName*. Setting the -hard argument to 0 lets the trace group be used to route any net; setting the argument to 1 (the default) routes only those nets with a matching function through the trace group.

**-from** *binName*
Makes the trace group directional. The trace group can only be used by nets with sources in the defined from bin.

## Example

```
trace_group_attribute MyTraceGroup -function GCLK

foreach tg [tss_list_traces -connections {mb.uA mb.uD}]
{trace_group_attribute $tg -connection_model DIRECT}
```

## Errors

An error is returned in these cases:

- hard or soft is used without a -function argument
- *trace_group_name* cannot be found

- *modelName* is not defined
- *binName* cannot be found
- A trace has more than one -from specification

# net_route

Writes detailed constraints to reduce unnecessary routing overhead during incremental runs with limited RTL changes. The command can constrain the following:

- A net to a trace for simple direct connections (trace assignment)
- Multiple nets to a trace using a TDM module (TDM assignment/trace assignment)
- Complex net routing (split/feedthrough nets) with trace and TDM module constraining for each segment of the route

## Syntax

**net_route** *netName*
**-from** *binName* **-to {***binNameList***}** [**-through {***binNameList***}**]
**-using_trace {***traceNameList***}**
[**-tdm_module {***moduleList***}**|**-tdm_inst_name** *instName* |
**-tdm_net_name** *netName*]

*netName*
The name of the net to be constrained.

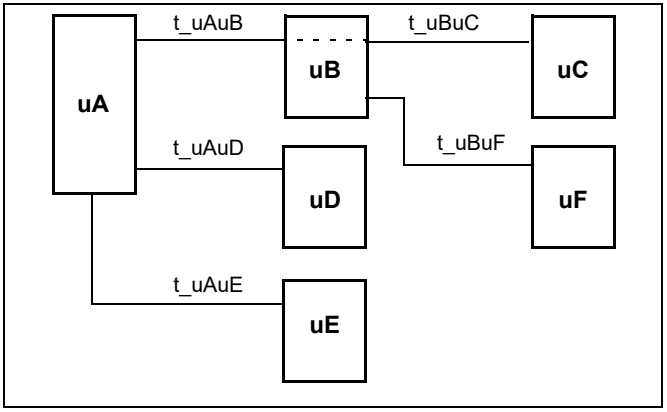
**-from** *binName* **-to {***binNameList***}** [**-through {***binNameList***}**]
Defines the origin (**-from**) and destination (**-to**) bins for the specified net. The optional **-through** argument specifies a feedthrough path.

**-using_trace {***traceNameList***}**
Identifies the board trace for the routed net. Multiple trace names indicate a feedthrough net.

**-tdm_module {***moduleList***}**|**-tdm_inst_name** *instName* |**-tdm_net_name** *netName*
Optional arguments to identify TDM modules associated with the routed net.

## Example

The following figure illustrates a complex route example for a multi-terminal net. The corresponding net_route command is included below.




```
net_route net
   -from uA -to uC -through {uB} -using_trace {t_uAuB t_uBuC}
      -tdm_module {DIRECT HSTDM4by2}
   -from uA -to uD -using_trace t_uAuD -tdm_module HSTDM8by2
   -from uA -to uE -using_trace t_uAuE -tdm_module HSTDM8by2
   -from uA -to uF -through {uB} -using_trace {t_uAuB t_uBuF}
      -tdm_module {DIRECT HSTDM4by2}
```

# Interactive Partitioning Commands

These commands are used as manual commands in an interactive partitioning session, or are entered into an interactive partitioning file that can be specified with the -ipcf argument when partitioning is run. See *Using Interactive Partitioning*, on page 355 in the *User Guide* for usage information.

## Interactive Partitioning Commands by Function

| | |
|---|---|
| **Flow Commands** | run_flow<br>run_route<br>rrun_write_results<br>exit |
| **Partitioning Commands** | assign_cell {cell(s)} {bin}<br>assign_port {port(s)} {bin}<br>replicate_cell {cell(s)} {bins}<br>fix_multihop_path -net name |
| **List Commands** | design_list_*<br>tss_list_* |
| **Reporting Commands** | report_cell_connections<br>report_solution<br>report_cost<br>report_interconnection_table<br>report timing options |
| **Solution Commands** | solution_commit<br>solution_discard<br>solution_save {file}<br>solution_restore {file} |

## Alphabetical List of Interactive Partitioning Commands

| Command | Descripton |
|---|---|
| assign_cell {*cell(s)*} {*bin*} | Assigns one or more cells to an FPGA bin in the temporary solution. |
| assign_port {*port(s)*} {*bin*} | Assigns one or more ports to a port bin in the temporary solution. |
| design_list_* | Lists properties for the specified object. For details, check the command using -help. For example: design_list_cells -help |

| Command | Descripton |
|---|---|
| exit | Ends the interactive session. It run_write_results was called successfully, it returns success. |
| fix_multihop_path -net name | Runs partitioning to fix the worst multi-hop for the given net. |
| replicate_cell {*cell(s)*} {*bins*} | Replicates one or more cells in multiple FPGA bins in the temporary solution. |
| report timing *options* | Reports timing.<br>• -type hops \| slack<br>The default is hops, which reports paths with the worst hops. Use slack to report paths with the worst slack.<br>• -worst_paths *value*<br>Reports paths with the worst timing.<br>• -level *value*<br>Reports multi-hop paths of the specified length.<br>• -net *value*<br>Reports the worst paths for the specified net.<br>• -summary *value*<br>Prints a summary.<br>• -file *value*<br>Writes the report to the specified file. |
| report_cell_connections | Reports cell connections, as in partition.rpt. |
| report_cost | Prints information about interconnections and their TDM requirements and is saved to a csv file. |
| report_interconnection_table | Prints information about interconnections and their TDM requirements, and is saved to a csv file. |
| report_solution | Reports bin usage (AP140). |
| run_flow | Runs the partition algorithm and switches to manual partitioning. |
| run_route | Runs the global router and prints a summary (AP267). |
| run_write_results | Runs global route and generates partitions, required to complete the flow. |
| solution_commit | Accepts all pending moves and applies them to the final solution. |
| solution_discard | Discards all pending moves. |

| Command | Descripton |
|---|---|
| solution_restore {*file*} | Restores the last saved solution from the file. |
| solution_save {*file*} | Saves the temporary solution to the specified file. |
| tss_list_* | Lists properties for the object specified in the command. For details, check the command using -help. For example: design_list_cells -help. |

SYNOPSYS®
Silicon to Software™

CHAPTER 4

# Target System Specification Commands

The ProtoCompiler tool lets you define the hardware setup using enhanced Tcl Commands. This chapter describes the Tcl commands used to define the hardware in target system specification (TSS) files. The TSS file is used by the partitioner.

At the early stages of design, the TSS file contains basic information and is used in conjunction with abstract commands in the pcf file to explore interconnect schemes and partition options. As the design is refined, the abstract pcf commands must be translated into specifications in the TSS file. You can only create partitions and system route the design with a fully-specified TSS file and pcf files that no longer contain any of the abstract commands.

For more information, see the following:

# HAPS Board Systems and the TSS File

Board files are automatically written based on user requirements specified using Tcl commands. The TSS file lets you specify the following:

- The systems and daughter boards used in the target system, and their setup
- The interconnect between boards, either by specifying the exact interconnect boards to be instantiated on the HAPS system, or by specifying trace requirements between FPGAs and allowing the tool to select the best configuration
- Board parameters such as voltage regions and clock configuration

For information about creating a TSS file and defining the hardware using the commands, see *Defining the System in a TSS File*, on page 184 in the *User Guide*. For details about the syntax of the commands used in the file, see *TSS Board System Commands*, on page 244.

# TSS Board System Commands

To access the command line help for the commands in the table, first open a TSS shell with the launch tss command (*launch tss*, on page 74) and then use the help command in this window.

These are the commands for the system:

| | | |
|---|---|---|
| board_system_configure | board_system_create | board_system_list |
| board_system_save | report_board_system | |

# board_system_configure

Defines the HAPS system clock and voltage-region parameters in a single- or multiple-board system.

## Syntax

**board_system_configure**
**-cde_chain** *chainList*
**-clk_gen -frequency** *value* **-source** *inputSource clkgeneratorName*
**-clk_src -frequency** *value* [**-name** *clkName*] *clkSourceName*
**-clock** *signalList Value*
**-load** *all | listOfFPGAs*
**-source** *fpgeName*
**-connect {***conn1 conn2***}**
**-connectGCLKS {***conn1 conn2***}**
**-connect_clocks -source** *inputSource clkList*
**-connector** [**-dci** *dciValue*] [**-voltage** *vltValue*] [**-vref** *vrefValue*] *connectorList*
**-create_ccm_clocks**
**-dci** *connectorList dciValue*
**-disconnect {***conn1 conn2***}**
**-ecdb** [**-a_clk_in** *clkValue*] [**-a_clk_out** *clkValue*] [**-b_clk_in** *clkValue*] [**-b_clk_out** *clkValue*] [**-c_clk_in** *clkValue*] [**-c_clk_out** *clkValue*] [**-d_clk_in** *clkValue*] [**-d_clk_out** *clkValue*] *ecdbName*
**-fpga_file** *deviceName deviceValue*
**-fpga_id** *deviceName deviceValue*
**-gpio** *gpioConnectors*
**-pll -frequency** *value* [**-name** *nameValue*] *pllName*
**-refclock** [**-frequency** *value*] **-name** *nameValue* **-source** *refclkSource refClkConnectors*
**-reset** *signalList signalValue*
**-single_fpga_target** *targetFPGA*
**-source_frequency** *BaseClockFrequencyInkHz*
**-top_io {***conn1 conn2***}**
**-voltage {***regionList***}** *voltageLevel*
**-vref** *connectorList connectorValue*

**-clk_gen -frequency** *value* **-source** *inputSource pllName*

Specifies the clock generators (PLL1/PLL2) with their input source and input frequency. This option is similar to the cfg_clock_set_input configuration command. The default value for -frequency is kHz. To specify it in MHz, add the letter m|M: -frequency 12m or -frequency 10M.

**-clk_src -frequency** *value* [**-name** *pllInputName*] *clkSourceName*
Creates and names clock sources.

**-clock** *signalList Value*
Identifies the HAPS clock sources. ***signalList*** is the list of clock signals, for example: GCLK0, GCLK1. ***Value*** is the driving signal, for example: fpga_a, pll1.

**-pll** *pllName* **-frequency** *value*
Specifies the PLL frequency. On HAPS-70 and HAPS-80 systems, it sets the frequency for PLL[1-2]_[1-4]; for HAPS-DX systems, it sets the frequency for PLL1_[1-4]. This option is like the cfg_clock_set_frequency configuration command. The default value for -frequency is kHz, but you can alternatively specify it in MHz, by adding the letter m|M to the value.

**-reset {***resetList***} 0|1**
Defines the reset network among FPGAs.

**-voltage {***regionList* | *conn***}** *voltageLevel*
Configures a board for different voltage levels. For HAPS-80D, use connector instead of region list.

**-connect {***conn1 conn2***}**
Specifies the UMRBus-to-system or system-to-system CDE/CLK pairs to be connected.

**-disconnect {***conn1 conn2***}**
Specifies the UMRBus-to-system or system-to-system CDE/CLK pairs to be disconnected.

**-top_io {***conn1 conn2***}**
Instantiates top-I/O boards at specified connectors as port bins.

**-single_fpga_target** *targetFPGA*
Specifies usage of a single FPGA for flow. This argument defines a single FPGA as the target for a multiple FPGA design to be implemented to.

**-connectors** {**-instance** *device*} {**-device** *device*} {**-available**} {**-occupied**}
Lists the connectors that are connected to the specified device.

**-create_ccm_clocks**
(*HAPS-100 only*) Creates controlled clocks using HAPS Clock Generator. The Clock Generator (HAPS Clockgen) provides emulation style clocking support to prototyping designs through the ProtoCompiler tool. Using HAPS Clockgen, clocks can be turned into controlled clocks that can be programmed (period, rise time, and fall time) using the Protocompiler

RunTime.
Only options allowed with this command are **-source**, **-load and -source_frequency**.

**-function** *listOfFunction* [*value*]
Configures interface panel function. *listOfFunction* is the list of panel function names, for example mictor or pmod. *value* is the panel function value, such as true, false, or xyz. (HAPS-80D only)

**-load** *all | listOfFPGAs*
(*HAPS-100 only*) List of FPGAs where the HAPS Clockgen load module is replicated.

**-source** *fpgeName*
(*HAPS-100 only*) FPGA on which the HAPS Clockgen source module is placed.

**-connectGCLKs** *clockList*
Connects two or more global clocks across systems.

**-name** *pllInputName*
Assigns a user-defined name to a PLL input clock source in the existing PLL. The name can then be used in a PCF constraint to assign a clock source destination.

**-source_frequency** *sourceClockFrequencyInkHz*
(*HAPS-100 only*) Configures the PLL clock frequency of the source clock in kilohertz (kHz).

## Examples

```
board_system_configure -clock {fb_1.GCLK1} PLL1

board_system_configure -voltage {fb_1.V1a fb_1.V2b} 2.5

//HAPS-80D only
board_system_configure -voltage {fb1.A4 fb1.B1} 1.00

board_system_configure -reset {fb1.CDE_OUT fb2.CDE_IN}

board_system_configure -connectGCLKs {fb1.GCLK1 fb2.GCLK2
fb3.GCLK3}
```

## HAPS Clockgen Example

```
//HAPS-100 only
board_system_configure -create_ccm_clocks -source mb1.uA -load
{mb1.uA mb1.uB}
```

## PLL Configuration Examples

```
// Sets input source to PLLIN. Changes PLL1_1 output frequency
// to 10M but other PLLs(PLL1_2,PLL1_3,PLL1_4) keep default value.
board_system_configure -clk_gen fb1.PLL1 -source PLLIN -frequency
10000

// Sets input source for PLL2 as GCLK0 and frequency to 12.30MHz
board_system_configure -clk_gen fb1.PLL2 -source GCLK0 -frequency
12.30M

// Sets PLL1_1 output freq to 30MHz; this also changes the input
// frequency of PLL1 to 30M
board_system_configure -pll fb1.PLL1_1 -frequency 30M

// Sets PLL1_2 output freq as 10MHz. No change in PLL1 input freq
board_system_configure -pll fb1.PLL1_2 -frequency 10M

// Sets PLL2_1 output freq as 40MHz. No change in PLL2 input
// frequency because it is driven by GCLK0.
board_system_configure -pll fb1.PLL2_1 -frequency 40M
```

## UMRBus Chaining Example

Use board_system_configure to specify the UMRBus chaining between HAPS systems. UMRBus chaining instantiates a CDE_CABLE across connectors.

```
board_system_create -add HAPS70_S48 -name fb1
board_system_create -add HAPS70_S24 -name fb2
board_system_configure -cde_chain {umrbus_if.CDE_OUT fb1.CDE_IN}
board_system_configure -cde_chain {fb1.CDE_OUT fb2.CDE_IN}
```

# board_system_create

Defines the board configuration in a multi-board system.

## Syntax

**board_system_create**
**-haps -name** *systemName*
**-add** *hapsBoard* **-name** *boardName* [**-stack_through** *value*]
**-stack {***boardInstNames***}** [**-skew {***row column***}** | **-connector {***pairList***}**]
**-speed_grade {***speedGrade***}**
**-interconnect -auto -width {***traceWidth***} -devices {***deviceList***}** |
**-manual** *interconnBoard* **-name** *connName* **-connector {***listofConnPairs***}**
**-connect_at top**|**bottom -hard 0**|**1**
**-disconnect** [*objectName*] [**-width** *value*] [**-devices** *value*]
**-haps_custom_db** *daughterboard* **-guts** *gutsFile*
[**-HSIO** *connectors*][**-HT2** *connectors*][**-HT3** *connectors*]
[**-MGB** *connectors*] [**-MGB2** *connectors*] [**-MGBFLEX** *connectors*]
[**-QSFP** *connectors*] [**-SATA** *connectors*]

**-clear_all**

**-haps -name** *systemName*
Creates a HAPS system. The -name argument specifies the HAPS system name.

**-add** *hapsBoard* **-name** *boardName* [**-stack_through** *value*]
Adds a HAPS system with the board name specified by the -name argument. Use the -objects option to the board_system_list command to list the valid HAPS object names for the *hapsBoard* argument. For example:

```
board_system_create -add HAPS100_4F -name FB1 -speed_grade {-1}
```

**-stack {***BoardInstNames***}** [**-skew {***row column***}** | **-connector {***pairList***}**]
Defines board stacking when multiple HAPS boards are included in the system. Optional offset stacking is accomplished either by defining the row-column skew between two boards (-skew argument) or by providing a list of connector pairs to be connected between two boards (-connector argument).

**-speed_grade {***speed***}**

Specifies the speed grade for all the FPGAs on the specified HAPS system being defined. For single-FPGA designs, set the speed grade using the option set command (*option*, on page 81). It is recommended that the speed grade be defined every time a system is added.

Speed grades vary with the technology. To find the valid speed grades for a selected technology, use the option get valid_speed_grades command. Different systems can have different speed grades.

Some supported speed grades are -1-c, -2-e, -1-c-es2, -2-e-es2. Here are some examples:

```
# Sets default on all FPGAs to -1-e:
board_system_create -add HAPS100_4F -name FB1 -speed_grade {-1}

# Sets default on all FPGAs to -2-e:
board_system_create -add HAPS100_4F -name FB1 -speed_grade {-2}
```

HAPS-100 4F is a mixed speed grade system. If you want high speed applications specify -M as the speed grade. FPGA A is given -2 speed grade, while the remaining FPGAs are given -1-e.

```
# Sets FPGA A to -2-e; default -1-e for the others:
board_system_create -add HAPS100_4F -name FB1 -speed_grade {-M}
```

**-interconnect -auto -width {***traceWidth***} -devices {***deviceList***} |**
**-manual** *interconnBoard* **-name** *connName* **-connector {***listofConnPairs***}**
**-connect_at top|bottom -hard 0 | 1**

Specifies the interconnections among devices. Use it to instantiate daughter boards. See *Daughter Board Instantiation*, on page 252 and *MGB Interface Card Instantiation Example*, on page 253. For further detail and examples, see *Defining Connections with board_system_create -interconnect*, on page 200 in the *User Guide*.

When the -auto argument is used, only the number of connections and the devices to be connected are required. The tool automatically instantiates interconnect boards to satisfy the requested number of connections. See Defining the Target System in a Detailed TSS File, on page 189 in the User Guide for information about how to use this argument to specify a number of traces, without specifying particular cable connections.

When the -manual option is used, the interconnect board (-name) and the list of board connectors where the interconnect board is connected (-connector) must be specified.

The -connect_at option is used with interconnect boards to indicate if the connection is through the top or bottom connectors.

The -hard option is used to set hard constraint on interconnect cable so that it cannot be modified or removed during auto cabling. The default is 0.

**-disconnect** [*objectName*] [**-width** *value*] [**-devices** *value*]
Removes the board, daughter board, or interconnect board from the system.

**-haps_custom_db** *daughterboard* **-guts** *gutsFile* [**-HT2** *connectors*] [**-HT3** *connectors*] [**-HSIO** *connectors*] [**-MGB** *connectors*] [**-MGB2** *connectors*] [**-MGBFLEX** *connectors*] [**-QSFP** *connectors*] [**-SATA** *connectors*]

Enables the addition of custom daughter boards to HAPS systems.

*daughterboard*
Sets the name for the daughter board. The same name is used during instantiation of the daughter board.

**-guts** *gutsFile*
Sets the logic file to use for the custom daughter board. Use this to provide the connectivity, trace names, constraints, and any other property to be attached to the database.

The -HT2, -HT3, -HSIO, -MGB, -MGB2, -MGBFLEX, -SATA, and -QSFP options are used to provide the name for the daughter board connectors respectively in the database.

**-clear_all**
Clears/resets the complete system.

## Examples

```
board_system_create -add HAPS100_4F -name FB1 -speed_grade {-2}

board_system_create -add HAPS100_4F -name FB1 -speed_grade {-M}

board_system_create -haps -name chip_proto_2

board_system_create -add HAPS70_S48 -name fb_1

board_system_create -stack {fb_1 fb_2} -skew {5 3}

board_system_create -interconnect -auto -width 120
   -devices {fb_1.uA fb_1.uB fb_1.uD}
```

```
board_system_create -haps_custom_db DB1_HT3 -HT3 {JX1 JX2 JX3}
   -guts {/nfs/test_fpga/db1/db1_guts.v} board_system_create
   -haps_custom_db DB2_MGB -MGB {JM1 JM2 JM3} -guts
   {/nfs/test_fpga/db2/db2_guts.v} board_system_create
   -haps_custom_db DB3_HT3_HSIO -HT3 {JX1 JX2 JX3} -HSIO {JH1}
   -guts{/nfs/test_fpga/db3/db3_guts.v}
```

## Interconnect Board Instantiation

The -interconnect option for the board_system_create command is used to instantiate interconnect boards such as an FMC_ADAPTER_HT3. In the following examples, an FMC_ADAPTER_HT3 is added to both a HAPS-70 S24 system and a HAPS-DX7 S4 system.

```
board_system_create -add HAPS70_S24 -name fb1
board_system_create -interconnect -manual FMC_ADAPTER_HT3
   -name fmc3 -connector {0 fb1.A3 fb1.A4 fb1.A5 fb1.A6}

board_system_create -add HAPSDX7_S4 -name fb
board_system_create -interconnect -manual FMC_ADAPTER_HT3
   -name fmc1 -connector {fb.AH1 fb.A3 fb.A4 fb.A5 fb.A6}
board_system_create -interconnect -manual FMC_ADAPTER_HT3
   -name fmc2 -connector {fb.AH2 fb.A7 fb.A8 fb.A9 fb.A10}
```

See Defining Connections with board_system_create -interconnect, on page 200 in the *User Guide* for further details and examples.

## Daughter Board Instantiation

The -interconnect option for the board_system_create command is also used to instantiate HAPS daughter boards.

In the following command example, a DDR3_SODIMM2R_HT3 daughter board is added to the target system.

```
board_system_create -interconnect -manual DDR3_SODIMM2R_HT3
   -name sram1 -connector {fb_1.A1}
```

Refer to Defining Connections with board_system_create -interconnect, on page 200 in the *User Guide* for more examples.

Use the board_system_list command to see a list of supported daughter boards, or refer to *Supported Board Types*, on page 259.

## MGB Interface Card Instantiation Example

MGB cards can be instantiated across MGB connectors available on the HAPS systems. The following sequence of board_system_create commands instantiates RISER1_MGB on HAPS system connector AM1 (connector MGB1 on FPGA A) and then adds a PCIE4_MGB board to riser connector J2 and a SATA4_MGB board to riser connector J3.

```
board_system_create -interconnect -manual RISER1_MGB
   -name conn1 -connector {0 0 fb1.AM1}

board_system_create -interconnect -manual PCIE4_MGB
   -name conn2 -connector {conn1.J2}

board_system_create -interconnect -manual SATA4_MGB
   -name conn3 -connector {conn2.J3}
```

Note that MGB cards cannot be attached directly to HAPS-100 modules see *Using MGB Interface Cards with HAPS-100 Modules*, on page 215 in the *User Guide*.

Use the board_system_list command to see a list of supported daughter boards and interface cards.

# board_system_list

Defines the board configuration in a multiple-board (cloud) system.

## Syntax

**board_system_list**
**-function -instance** *boardInstance*
**-objects** [**-type** *boardType*] board
**-instances** [**-type** *boardType*] |
**-type_of** *instanceType* |
**-value -parameter** *parameterValue* |
**-connectors** [**-instance** *value*] [**-device** *value*][**-available**] [**-occupied**]
**-parameters -instance** *boardInstName* [**-type** *parameterType*]

**-function -instance** *boardInstance*
Instance of the board.

**-objects** [**-type** *boardType*]
Lists all the supported board type objects. The optional -type argument filters objects based on their *boardType* (motherboard, interconnectboard, or daughterboard).

**-instances** [**-type** *boardType*]
Lists all the instances included in the system. The optional -type argument filters instances based on their *boardType* (motherboard, interconnectboard, or daughterboard).

**-type_of** *instanceType*
Lists the object type of the instance.

**-value -parameter** *parameterValue*
Lists the value of the specified parameter.

**-connectors** [**-instance** *value*] [**-available**] [**-occupied**]
Lists board connectors and their status. Use the argument **-instance** to filter by board instances, **-available** to filter by available connectors, or **-occupied** to filter by occupied connectors.

**-parameters -instance** *boardInstName* [**-type** *parameterType*]
Lists all the clock/PLL and voltage parameters for the specified board instance. The optional -type argument filters objects based on their *parameterValue* (clock or voltage).

## Description

The board_system_list command describes the configuration of the board system. The -objects option lists the supported systems, daughter boards, and interconnect boards according to the -type argument.

The remaining board_system_list command options list the actual HAPS system configuration and report an error if a HAPS system has not been defined with a previous board_sytem_create command.

## Examples

```
board_system_list -objects

board_system_list -objects -type motherboard

board_system_list -instances -type motherboard

board_system_list -parameters -instance fb1

board_system_list -parameters -instance mb1 -type clock
```

# board_system_save

Saves a board definition to the specified file. The board_system_save command is not generally required because the board definition file is automatically saved, but you can use it to save incremental changes during board file development or to write out a clean TSS.

## Syntax

**board_system_save**
**-board** *fileName*.**tss** [**-hstdm**]
**-tcl** *filename*
**-tsd** *filename*

**-board** *fileName*.**tss** [**-hstdm**] |
Creates the specified board file for the board system. If the optional -hstdm argument is specified, generates board file with only CON_CABLE connections for the connectors between devices for HSTDM compatibility (HSTDM requires CON_CABLE connections exclusively).

**-tcl** *filename*
Writes out a file with the TSS information. The file includes all TSS commands from the TSS file with Tcl variables expanded, and excludes any other Tcl constructs the original input TSS contained. Use this option to write out a clean TSS.

**-tsd** *filename*
Writes out a file with the TSD information.

## Examples

```
board_system_save -board big_chip_system.tss
```

# report_board_system

Writes a detailed board-system report to the specified file that outlines the complete board-system configuration. If a path is not specified as part of the file name, file will be saved in the prepartition directory.

**report_board_system** *fileName*.**txt** [**-hstdm**]

*fileName.txt*
Name of the report file.

**-hstdm**
Consider CABLE interconnects for TDM.

SYNOPSYS®
Silicon to Software™

CHAPTER 5

# UPF Commands

This chapter describes the supported IEEE Unified Power Format (UPF) commands. For the standard compiler, the commands are specified in a UPF script file at the run pre_map stage. For the UC flow, the UPF commands are specified as part of VCS compile (see *Preparing the Input for Unified Compile, on page 117* in the *User Guide*).

- UPF Command Support
  The tool supports these commands for isolation, retention, power domains, and scope resolution.

| | |
|---|---|
| associate_supply_set | set_domain_supply_net |
| corrupt_pd | set_equivalent |
| create_power_domain | set_isolation |
| create_power_switch | set_isolation_control |
| create_supply_set | set_retention |
| load_upf | set_retention_control |
| map_retention_cell | set_scope |
| name_format | use_map_retention_cell |

- UPF Commands (UC Flow), on page 286

For information about how to use the commands, see See *Including UPF Specifications, on page 827* in the *User Guide*.

For information about support for UPF commands to run unified compile and the details about these commands, refer to *Using Unified Compile*, on page 100 in the *User Guide* and the VCS documentation, respectively.

# associate_supply_set

The associate_supply_set command associates a supply set handle with a supply set for a given power domain. The supply set handle defines the functionality for the supply net.

See *Defining Supply Sets, on page 843* and Defining the Power Domain Supply, on page 833 in the *User Guide* for more information.

## Syntax

**associate_supply_set** *supplySetRef* **-handle** *supplySetHandle*

*supplySetRef*
The root name of the associated supply set.

**-handle** *supplySetHandle*
Specifies the functionality for the associated supply net.

## Examples

```
create_power_domain PD1
create_power_domain PD2

create_supply_set ss1
create_supply_set ss2

associate_supply_set ss1 -handle PD1.primary
associate_supply_set ss2 -handle PD2.primary
```

# corrupt_pd

Controls the corruption of sequential elements (RAM or flip-flops) for specific power domains when it is shut-off.

## Syntax

**corrupt_pd -domain all** | *domainName* | *instanceName*
[**-mode** {**all0s** | **all1s** | **random** | **none**}]

**-domain**

Defines the power domains or instances for which corruption is applied.

- all
  Specifies all the switchable power domains to control corruption.
- *domainName*
  Specifies the name of the power domain to control corruption.
- *instanceName*
  Specifies the name of the instance to be ignored from corruption.

**-mode**

Implements sequential elements to one of the states below when the power domain is shut-off.

- all0s
  Corruption of sequential elements is set to all zeros. This is the default.
- all1s
  Corruption of sequential elements is set to all ones.
- random
  Corruption of sequential elements is randomly set to all0s and all1s.
- none
  No corruption occurs.

## Description

In the FPGA, when a power domain is powered off, the state of the domain becomes unknown or corrupted. UPF support implements isolation and retention logic so that the power domain is held to a known state during shut-off. Use the corrupt_pd command to specify how to implement corruption for instances or the sequential elements of specific power domains.

## Examples

Here are examples of various usage scenarios.

### Examples 1:

Enable all0s corruption for all domains globally.

```
corrupt_pd -domain all -mode all0s
```

Enable all1s corruption for power domain PD1.

```
corrupt_pd -domain PD1 -mode all1s
```

Ignore power domain PD2 from corruption.

```
corrupt_pd -domain PD2 -mode none
```

### Examples 2:

Enable all1s corruption for power domain PD1.

```
corrupt_pd -domain PD1 -mode all1s
```

Enable all0s corruption for power domain PD2.

```
corrupt_pd -domain PD2 -mode all0s
```

Enable random corruption for power domain PD3.

```
corrupt_pd -domain PD3 -mode random
```

### Examples 3:

Enable all0s corruption globally.

```
corrupt_pd -domain all -mode all1s
```

Ignore sequential instance inst from corruption.

```
corrupt_pd -domain {inst} -mode {none}
```

### Examples 4:

Enable all1s corruption for power domain PD1.

```
corrupt_pd -domain {PD1} -mode all1s
```

Ignore sequential instance u1/inst from corruption.

```
corrupt_pd -domain {u1/inst} -mode {none}
```

Note that warnings are generated for the following conditions:

- Corruption for the power domain will be ignored because the specified power domain does not exist.
- Corruption for the power domain will be ignored because the it does not include the specified switch, such as all1s.

# create_power_domain

Creates a power domain, which provides a power supply distribution network. For more information, see Specifying UPF Power Domains, on page 830 in the *User Guide*.

## Syntax

**create_power_domain** *domainName*
[**-elements {***list***}**]
[**-supply {***supplySetHandle* [*supplySetRef*]**}**]
[**-include_scope**]
[**-scope {***scopeInstance***}**]

*domainName*
Specifies the name of the power domain.

**-elements {***list***}**
Specifies a list of hierarchical design elements assigned to the power domain. Note that the -elements option entries can include the wildcard (*) character.

**-supply {***supplySetHandle* [*supplySetRef*]**}**
Specifies the supply set name associated with the supply set handle, where the specified supply set will be equivalent to the supply set handle. By default, the power domain uses the following supply net handles:

- primary – primary supply set for the power domain.
- default_isolation – default isolation supply set if isolation power is not defined.
- default_retention – default retention supply set if retention power is not defined.

**-include_scope**
Defines the extent of the power domain. This includes the active scope, and by default, all of its descendant scopes.

**-scope {***scopeInstance***}**
Defines the scope (hierarchical logic level) for the power domain that is created. By default, the power domain is created in the current scope.

## Description

The create_power_domain command creates a power domain in the specified scope. A power domain contains a collection of design elements that share a primary power net and ground power net. The scope of the power domain is the hierarchical logic level in which the power domain is created. The set of design elements belonging to the power domain is referred to as the extent of that power domain.

## Examples

```
create_power_domain PD_Top -include_scope

create_power_domain PD2 -scope U2 -elements {U1/U21 U2/U22}

create_power_domain PD1 -supply {primary ss1}
```

The following example shows how the wildcard (*) character can be used with the -elements option for this command.

```
create_power_domain PD -elements {
   inst_top/inst[*].gen/mod*
   inst_top/*/inst*
   inst_top/inst_mod/mod
}
```

# create_power_switch

The create_power_switch command, in combination with the corrupt_pd attribute setting, implements logic to force corruption of the sequential elements within a prototyping power domain when the domain is shut-off.

## Syntax

**create_power_switch** *switchName* **-domain** *domainName*
**-control_port {***portName netName***}** *
**-on_state {***stateName inputSupplyPort* **{***booleanExpression***}}** *
[**-off_state {***stateName* **{***booleanExpression***}}** ]
[**-ack_port {***portName netName* **{***booleanExpression***}}** ]

*switchName*
Name of the power switch for the power domain.

**-domain** *domainName*
Name of the power domain that contains the specified power switch.

**-control_port {***portName netName***}**
A control port on the power switch and the net to which the port is connected. The -control_port argument is required and can be repeated.

**-on_state {***stateName inputSupplyPort* **{***booleanExpression***}}**
Implements the power domain on-state for the input supply port defined by its corresponding Boolean expression. The -on_state argument is required and can be repeated.

**-off_state {***stateName* **{***booleanExpression***}}**
Optional argument that implements the power domain off-state defined by its corresponding Boolean expression.

**-ack_port {***portName netName* **{***booleanExpression***}}**
Specifies the name of the acknowledge port for the power switch and the logical net to which this port is connected. Optionally, you can specify a boolean expression for the specified control ports.

For the Boolean expression syntax in the above arguments, only the following operators (plus parentheses) can be used:

| Operator | Description |
|---|---|
| ~ & \| ^ | Bit-wise negation, AND, OR, XOR |
| ! && \|\| | Logical negation, AND, OR |

## Description

In actual ASICs, when a power domain is powered off, the state of the domain becomes unknown or corrupted. When isolation and retention strategies are employed, the state of the power domain is restored on power up. With FPGA prototyping, UPF support implements isolation and retention logic so that the power domain is held to a known state during shut-off. The on/off state of the power domain is determined using the create_power_switch command, and corruption is enabled by setting the corrupt_pd attribute that implements logic to force the sequential elements to one of the below states when the domain is shut-off:

- All ones
- All zeros
- Random ones and zeros

Adding power domain corruption to FPGA prototyping increases coverage and adds confidence that the design will continue to function reliably when the ASIC is fabricated. For more information, see corrupt_pd, on page 555.

Additionally, the power switch drives the acknowledge port with the control ports based on the on_state/off_state condition or the boolean expression specified.

## Example

```
create_power_switch sw1 \
   -domain PD1 \
   -control_port {ctrl_small ON1} \
   -control_port {ctrl_large ON2} \
   -control_port {ss SUPPLY_SELECT} \
   -on_state {full_s1 vin1 {ctrl_small & ctrl_large & ss}} \
   -off_state {not_required {!ctrl_small & !ctrl_large}} \
   -ack_port {ack_p ACKN {ctrl_small & !ctrl_large}}
```

## Restrictions

Control port names cannot conflict with SystemVerilog keywords.

# create_supply_set

The create_supply_set command explicitly creates supply sets. The supply set can be used to provide more flexibility for isolating cells in a power domain when using the -source/-sink/-diff_supply_only arguments for the set_isolation command. To implicitly create supply sets, see *create_power_domain*, on page 265.

For more information about using supply sets, see Defining Supply Sets, on page 843 in the *User Guide*.

## Syntax

**create_supply_set** *supplySetName*
[**-function {power** *netName***} -function {ground** *netName***}**]
[**-update**]

*supplySetName*
The name of the supply set.

**-function {power** *netName***} -function {ground** *netName***}**
A power/ground pair that specifies the functionality for the corresponding supply net (*netName*). The supported function arguments are power and ground.

**-update**
Updates an existing supply set with the specified supply sets in the command.

## Example 1

```
create_supply_set ss1
create_supply_set ss2

set_domain_supply_net PD1 -primary_power_net ss1.power
   -primary_ground_net ss1.ground
set_domain_supply_net PD2 -primary_power_net ss2.power
   -primary_ground_net ss2.ground

create_supply_set ss1 -function {power VDD}
   -function {ground GND} -update
create_supply_set ss2 -function {power VDD1}
   -function {ground GND} -update
```


## Example 2

```
create_supply_set ss1
create_supply_set ss2

set_domain_supply_net PD1 -primary_power_net ss1.power
   -primary_ground_net ss1.ground
set_domain_supply_net PD2 -primary_power_net ss2.power
   -primary_ground_net ss2.ground

create_supply_set ss1 -function {power VDD}
   -function {ground GND} -update
create_supply_set ss2 -function {power ss1.power}
   -function {ground ss1.ground} -update
```

# load_upf

Loads the script file that executes the UPF commands for the defined scope. For more information, see Specifying UPF Power Domains, on page 830 in the User Guide.

## Syntax

**load_upf** *upfFilename* [**-scope** *instanceName* ]

*upfFilename*
Identifies the script file containing the UPF commands.

**-scope** *instanceName*
Sets the scope for the UPF commands. The scope defines the hierarchical instance to which the UPF commands are applied.

# map_retention_cell

The map_retention_cell command lets you customize retention for sequential elements. To specify how the map_retention_cell command can be implemented use the set_option command, see option, on page 81For information about specifying UPF retention logic, see *Modeling Retention Logic*, on page 847 of the *User Guide*.

## Syntax

**map_retention_cell** *retentionName* **-domain** *domainName* [**-elements {***list***}**] [**-lib_cells {***list***}**] [**-lib_cell_type** *libCellType*][**-enable_upf_map_retention_cell 0 | 1**]

**map_retention_cell** *retentionName*
Name of the retention strategy for the specified domain.

**-domain** *domainName*
Specifies the power domain for which the retention strategy is applied.

**-elements {***list***}**
Specifies a list of design elements, sequential registers, or signals of sequential elements to be mapped for retention.

**-lib_cells {***list***}**
Specifies library cells for the elements to be mapped for retention.

**-lib_cell_type** *libCellType*
Specifies the type of library cells identified by the retention attribute, which has the same retention behavior as that of the inferred RTL for the sequential elements.

**-enable_upf_map_retention_cell 0 | 1**
Overrides default retention model.

## Examples

```
set_retention RET -domain PD -elements {e1 e2}
map_retention_cell RET -domain PD -lib_cells RET_FF

set_retention RET -domain PD -elements {e1 e2}
map_retention_cell RET -domain PD -lib_cell_type RET_FF_TYPE

set_retention RET -domain PD -elements {e1 e2}
map_retention_cell RET -domain PD -lib_cells RET_FF
   -lib_cell_type RET_FF_TYPE
```

# name_format

Sets the isolation cell naming for UPF commands.

## Syntax

**name_format**
[**-isolation_prefix** *prefixName*]
[**-isolation_suffix** *suffixName*]

**-isolation_prefix** *prefixName*
Sets the prefix for isolation cell names. The default is "" (no prefix).

**-isolation_suffix** *suffixName*
Sets the suffix for isolation cell names. The default is "_UPF_ISO".

# set_domain_supply_net

The set_domain_supply_net command specifies the primary supply set for the power domain. See *Specifying UPF Power Domains, on page 830* in the *User Guide* for usage information.

## Syntax

**set_domain_supply_net** *domainName*
**-primary_power_net** *supplyNetName* **-primary_ground_net** *supplyNetName*

*domainName*
The domain name for which the default supply nets are applied.

**-primary_power_net** *supplyNetName*
Specifies the primary power supply net.

**-primary_ground_net** *supplyNetName*
Specifies the primary ground net.

## Example 1

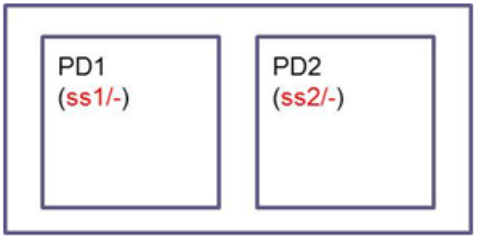
```
set_domain_supply_net PD1 -primary_power_net VDD
   -primary_ground_net GND

set_domain_supply_net PD2 -primary_power_net VDD1
   -primary_ground_net GND
```


## Example 2

```
create_supply_set ss1
create_supply_set ss2
```

```
set_domain_supply_net PD1 -primary_power_net ss1.power
   -primary_ground_net ss1.ground
set_domain_supply_net PD2 -primary_power_net ss2.power
   -primary_ground_net ss2.ground
```



# set_equivalent

The set_equivalent command allows you to specify two supply nets or supply sets to be equivalent.

## Syntax

**set_equivalent -nets** [*supplyNets* |*supplyPorts*] |**-sets** [*supplySets*|*supplySetHandles*] ]

The following table describes the command arguments and options. Note that the -net and -set options are mutually exclusive.

| Option | Description |
|---|---|
| **-nets** [*supplyNets* |*supplyPorts*] | Specifies the supply nets to be equivalent. When supply ports are specified for the argument, equivalence is applied to the supply nets connected to the port. |
| **-sets** [*supplySets* |*supplySetHandles*] | Specifies the supply sets to be equivalent. Equivalence is applied between the supply nets for the corresponding functions of the supply sets. |

## Description

The set_equivalent command enhances support for supply set association and can impact supply-dependent isolation strategies using the -source/-sink or -diff_supply_only options.

## Example

The set_equivalent command can be used as follows:

- Specify isolation strategy with -source/-sink.

  ```
  set_isolation iso_PD1 -domain PD1 -applies_to outputs -sink SS2
  ```

  Then, add the set_equivalent commands.

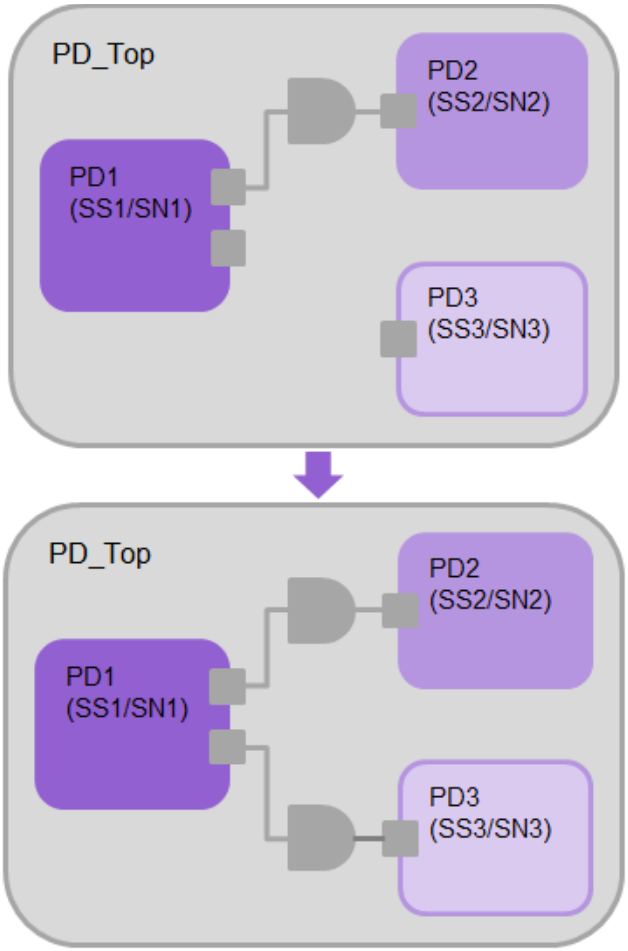
  ```
  set_equivalent -sets {SS2 SS3}
  set_equivalent -sets {SS2 PD3.primary}
  set_equivalent -nets {SN2 SN3}
  set_isolation iso_PD1 -domain PD1 -applies_to outputs -sink SS2
  ```

- Specify isolation strategy with -diff_supply_only.

```
set_isolation iso_PD1 -domain PD1 -applies_to outputs
   -diff_supply_only TRUE
```

Then, add the set_equivalent commands.

```
set_equivalent -sets {SS1 SS3}
set_equivalent -sets {PD1.primary SS3}
set_equivalent -nets {SN1 SN3}
set_isolation iso_PD1 -domain PD1 -applies_to outputs
   -diff_supply_only TRUE
```



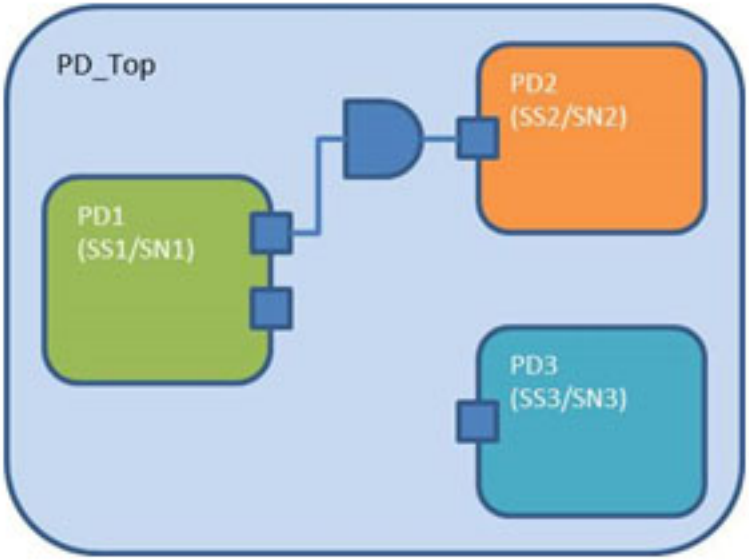
- Specify isolation strategy with -diff_supply_only.

```
set_isolation iso_PD1 -domain PD1 -applies_to outputs
   -diff_supply_only TRUE
```

Then, add the set_equivalent command.

```
set_equivalent -sets {SS1 SS3}
set_equivalent -sets {PD1.primary SS3}
set_equivalent -nets {SN1 SN3}
set_isolation iso_PD1 -domain PD1 -applies_to outputs
   -diff_supply_only TRUE
```

# set_isolation




Specifies the isolation strategy for a power domain and the elements in the domain with the applied strategy. An isolation strategy specification includes the enable signal net and clamp value, and specifies if inputs, outputs, or both inputs and outputs are isolated for the domain. All isolation strategies defined with this command, except those specifying no isolation, must have a corresponding set of control commands defined with the set_isolation_control command (see *set_isolation_control*, on page 282).

For additional information, see Specifying Isolation Cell Strategies, on page 835 in the *User Guide*.

## Syntax

**set_isolation** *strategyName* **-domain** *domainName*
[**-elements {***list***}**]
[**-applies_to inputs**|**outputs**|**both**]
[**-no_isolation**]
[**-clamp_value 0**|**1**|**latch**]
[**-source** *sourceSupplyRef*]
[**-sink** *sinkSupplyRef*]
[**-diff_supply_only true**|**false**]

*strategyName*
Specifies the isolation strategy name. The command must be specified with the same set_isolation_control strategy name.

**-domain** *domainName*
Specifies the power domain for which the isolation strategy is applied.

**-elements {***list***}**
Specifies a list of power domain boundary ports to which the isolation strategy is applied (overrides the -applies_to option).

**applies_to inputs**|**outputs**|**both**
Defines if input ports, output ports, or both input and output ports are isolated for the domain.

**-no_isolation**
Prohibits isolation from occurring for the power domain boundary ports specified in the isolation strategy.

**-clamp_value 0**|**1**|**latch**
Defines the clamp value of an isolated port for the isolation signal. The default is 0.

**-domain** *domainName*
Specifies the power domain for which the isolation strategy is applied.

**-source** *sourceSupplyRef*
Defines a name of a supply set reference for the drivers.

**-sink** *sinkSupplyRef*
Defines a name of a supply set reference for the receivers.

**-diff_supply_only true**|**false**
Introduces an isolation cell only if the driver and the receiver supply sets are different. The default value is FALSE.

## Using Tcl Variables with the set_isolation Command

The set_isolation command can be used with the following Tcl variables to control how the -applies_to option is implemented:

- enable_upf_1_0_applies_to_default_output – When the -applies_to option is not specified, use with the following syntax to control how the power domain is instrumented:

  **option set enable_upf_1_0_applies_to_default_output true|false**

- upf_iso_filter_elements_with_applies_to – When the -applies_to option is specified with the -elements option, use with the following syntax to filter elements based on the direction of the -applies_to for the power domain:

  **option set upf_iso_filter_elements_with_applies_to enable|disable|error**

Prior to UPF 1.0, the default isolation side (in the absence of -applies_to) was outputs which was changed to both beginning with the UPF2.0 standard. This Tcl variable provides backwards compatibility to revert the isolation to the output side for UPF1.0 strategies instead of isolating both sides.

For details on setting the option, see *option*, on page 81 in the *ProtoCompiler Command Reference*.

## Examples

```
set_isolation PD1_ret -domain PD1 -elements
   {U1/out_reg U2/out_reg}

set_isolation PD1_iso -domain PD1 -clamp_value 1
   -applies_to outputs

set_isolation PD1_no_iso -domain PD1 -no_isolation
   -elements {port1 port2}

set_isolation iso_PD1 -domain PD1 -applies_to both
   -diff_supply_only true
```

# set_isolation_control

Identifies the isolation strategy and specifies the isolation control signals for the strategy. A corresponding set of control commands for each isolation strategy must be defined with the set_isolation command (see *set_isolation*, on page 279). For additional information, see Specifying Isolation Cell Strategies, on page 835 in the *User Guide*.

## Syntax

**set_isolation_control** *strategyName* **-domain** *domainName*
**isolation_signal** *signalName*
[**-isolation_sense high|low** ]
[**-location self|parent|fanout]**

*strategyName*
Specifies the isolation strategy name. The set_isolation_control command must be specified with the same set_isolation strategy name.

**-domain** *domainName*
Specifies the power domain for which the isolation strategy is applied.

**-isolation_signal** *signalName*
Specifies the isolation signal name that causes the element for the applied isolation strategy within the domain to drive its clamp value.

**-isolation_sense high|low**
Specifies the logical sense for the isolation control signal. The default is high.

**-location self|parent|fanout**
Defines where the isolation cells are placed. The self option includes the isolation cells within the module being isolated, the parent option includes these cells in the parent of the module being isolated, and the fanout option defines where the isolation cells are placed in the logic hierarchy. The default is self.

## Examples

```
set_isolation_control ISO_on_outputs -domain PD1
   -isolation_signal iso_en1 -isolation_sense high
   -location self
```

# set_retention

Specifies the registers in the domain that are implemented as retention registers and identifies the save and restore signals for the retention functionality. Only registers in the elements list are given retention capabilities. For more information, see Modeling Retention Logic, on page 847 in the User Guide.

All isolation strategies defined with this command, except those specifying no retention, must have a corresponding set of control commands defined with the set_retention_control command (see *set_retention_control*, on page 284).

## Syntax

**set_retention** *strategyName* **-domain** *domainName*
[**-elements {***list***}**]
[**-no_retention**]

*strategyName*
Specifies the retention strategy name. The set_retention command must be specified with the same set_retention_control strategy name.

**-domain** *domainName*
Specifies the power domain for which the retention strategy is applied.

**-elements {***list***}**
Specifies a list of design or storage elements in the power domain (registers, RAMs, sequential shifters, FSMs) to which the retention strategy is applied.

**-no_retention**
Prevents the retention capability from being added to the elements identified by the retention strategy.

## Examples

```
set_retention PD1_ret -domain PD1 -elements
   {U1/out_reg U2/out_reg}

set_retention PD1_no_ret -domain PD1 -no_retention
   -elements {R1 R2 R3}
```

# set_retention_control

Specifies the retention control signal and its logical sense for the strategy. Alternatively, use the map_retention_cell command (*map_retention_cell*, on page 272) for user-defined retention schemes. Both of these commands require a corresponding control command to be set for each retention strategy (see *set_retention*, on page 283).

For usage information about UPF retention logic, see Modeling Retention Logic, on page 847 of the User Guide.

## Syntax

**set_retention_control** *strategyName* **-domain** *domainName*
**-save_signal {***logicNet* **high|low} -restore_signal {***logicNet* **high|low}**]

*strategyName*
Specifies the retention control strategy name. The set_retention_control command must be specified with the same set_retention strategy name.

**-domain** *domainName*
Specifies the power domain for which the retention control strategy is applied.

**-save_signal {***logicNet* **high|low}**
Specifies the signal (net/port/pin) that forces the register values to be saved into the shadow registers.

**-restore_signal {***logicNet* **high|low}**
Specifies the signal (net/port/pin) that forces the register values to be restored.

## Examples

```
set_retention_control PD1_ret -domain PD1 -save_signal
   {signal_net high} -restore_signal {signal_ret low}
```

# set_scope

Specifies the hierarchical scope within which the UPF commands are applied. For more information, see Specifying UPF Power Domains, on page 830 in the User Guide.

## Syntax

**set_scope** [*instanceName*]

*instanceName*

Specifies the instance name of the scope for which the UPF commands are applied. If the instance path:

- is not specified, the scope applies to the top-level current design.
- is specified with ".", the scope applies to the working instance.
- is specified with "..", the scope applies to the hierarchy one level up from the current design.
- is specified with "/", the scope applies to the working instance of the design after the /.
- specifies that multiple levels of hierarchy can be traversed in a single call to the set_scope command, separate the cell names with a slash (/).

# use_map_retention_cell

The use_map_retention_cell Tcl variable lets you determine how the tool uses the map_retention_cell command. For details about using the map_retention_cell command, see *map_retention_cell*, on page 272 and Modeling Retention Logic, on page 847 in the *User Guide*.

## Syntax

**use_map_retention_cell 0|1**

The following table describes the Tcl variable arguments.

| Option | Description |
|---|---|
| **use_map_retention_cell 0\|1** | You can set the use_map_retention_cell command to either:<br>• 0 – Parse and ignore the map_retention_cell command. This is the default.<br>• 1 – Process the map_retention_cell command. The tool looks for user-defined retention models and issues an error when they cannot be found. |

## Example

The use_map_retention_cell Tcl variable should be specified near the top of the UPF file so that it occurs before the first map_retention_cell command:

```
set use_map_retention_cell 0

set use_map_retention_cell 1
```

# UPF Commands (UC Flow)

The following UPF functionality is supported with the UC flow:

- Power domain
- Isolation (clamp 0/1/LATCH), Retention (1-pin, 2-pin)
- Power switch
- Supply sets, nets (always-on)

- Simstates
- Corruption (flops, memory, boundary mux)

## VCS Command Syntax for UPF

With the UC flow, the UPF file is sourced from the vcs command in the VCS command script. This is the UPF command:

**vcs -upf** <*designtop.upf*> **-power=unified_model -power_top** <*designtop*>
**[-power=scm_mem_ret]**
**[-power=seqcorrupt] [-power=hw_corrupt_memory]**
**[-power=enable_boundary_gates]**

**-upf** <*designtop.upf*>
Specifies the top-level UPF file.

**-power=unified_model**
Enables the retention of registers.

**-power_top** <*designtop*>
Use it to specify the top-level design only when the designtop.upf file does not contain the set_design_top *designtop* command.

**-power=scm_mem_ret**
Enables the retention of memory.

**-power=seqcorrupt**
Enables flop corruption. Then use the set_design_attributes -attribute SNPS_random_corruption command to determine the type of corruption:

0: All zeros corruption
1: All ones corruption
I: Inversion-based corruption

Note that corruption significantly increases the area by adding LUTs and registers.

**-power=hw_corrupt_memory**
Enables memory corruption. Then use the set_design_attributes -attribute SNPS_random_corruption command to determine the type of corruption:

0: All zeros corruption
1: All ones corruption
I: Inversion-based corruption

Note that corruption significantly increases the area by adding LUTs and registers.

**-power=enable_boundary_gates**
Enables boundary mux-based corruption.

## UPF Reports

In the UC flow, the UPF log information is written under the generated ucdb directory (ucdb/mvsim_native_reports). Check these files:

- isolation_assertion_summary.rpt
- isolation_association.rpt
- zebu_retention_flop_implementation.rpt
- upf_supply_port.rpt

SYNOPSYS®
Silicon to Software™

CHAPTER 6

# FDC Constraint Command Reference

This chapter describes timing and miscellaneous constraints specified with the following commands entered in an FDC constraint file that is then read by the run pre_map task. Commands can also be specified through a graphical constraint editor. Constraint commands are listed in the following tables.

## Timing Constraints

| | | |
|---|---|---|
| create_clock | set_clock_latency | set_min_delay |
| create_generated_clock | set_clock_uncertainty | set_multicycle_path |
| create_clockgen_group | set_datapathonly_delay | set_output_delay |
| reset_path | set_false_path | set_reg_input_delay |
| set_case_analysis | set_input_delay | set_reg_output_delay |
| set_clock_groups | set_max_delay | |

## Miscellaneous Constraints

| | |
|---|---|
| define_haps_fpga | define_io_standard |
| define_haps_io | reset_path |

# create_clock

Creates a clock object and defines its waveform in the current design.

## Syntax

**create_clock**
**-name** *clockName* [**-add**] **{***objectList***}**
**-period** *value*
[**-waveform {***riseValue fallValue***}**]
[**-disable**]
[**-comment** *commentString*]

**-name** *clockName* [**-add**] **{***objectList***}**
Specifies the name for the clock being created. If this argument is not used, the clock is assigned the name of the first clock source specified in *objectList*. If *objectList* is not used, the -name option must be specified, which creates a virtual clock with no associated port, pin, or net. Both -name and *objectList* give the clock a more descriptive name than the first source pin, port, or net. The -add option requires the -name option, and the clocks with the same source must have different names.

**-add**
Specifies if the clock is to be added to the existing clock or if it is to be overwrite the clock. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. The -name option is required.

**{***objectList***}**
Clocks can be defined on the following objects: pins, ports, and nets.

**-period** *value*
Specifies the clock period in nanoseconds. The *value* type must be greater than zero.

**-waveform {***riseValue fallValue***}**
Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise followed by fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform with a rise edge of 0.0 and a fall edge of *periodValue*/2 is assumed.

**-disable**
Disables the constraint.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

Refer to the following examples.

### Example 1

A clock named clk_in1 is created for port clk_in1 that uses a period of 10 with rising edge of 0 and falling edge of 5.

```
create_clock -name {clk_in1} -period 10 [get_ports {clk_in1}]
```

### Example 2

A clock named clk is created for port clk_in that uses a period of 10.0 with rising edge of 5.0 and falling edge of 9.5.

```
create_clock -name {clk} -period 10 -waveform {5.0 9.5}
   [get_ports {clk_in}]
```

### Example 3

A virtual clock named CLK is created that uses a period of 12 with a rising edge of 0.0 and falling edge of 6.0.

```
create_clock -name {CLK} -period 12
```

# create_generated_clock

Creates a generated clock object.

## Syntax

**create_generated_clock**
**-name** *clockName* [**-add**] | **{***clockObject***}**
**-source** *masterPinName*
[**-master_clock** *clockName*]
[**-divide_by** *integer* | **-multiply_by** *integer* [**-duty_cycle** *value*]]
[**-invert**]
[**-edges {***edgeList***}**]
[**-edge_shift {***edgeShiftList***}**]
[**-combinational**]
[**-disable**]
[**-comment** *commentString*]

**-name** *clockname*
Specifies a name for the generated clock. If this option is not used, the tool automatically assigns the clock the name of the first clock source specified in the -source option. If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.

Use the name option to rename automatically-generated clocks on clock blocks like MMCMs and PLLs.

**-add**
Specifies if the clock is to be added to the existing clock; if not, the clock is overwritten. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also use the -name and -master_clock options.

**{***clockObject***}**
The first clock source specified in the -source option in the absence of *clockName*. Clocks can be defined on pins, ports, and nets.

**-source** *masterPinName*
Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used to derive the generated clock waveform.

**-master_clock** *clockName*
Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.

**-divide_by** *integer*
Specifies the frequency division factor. If the divide factor value is 2, the generated clock period is twice as long as the master clock period.

**-multiply_by** *integer*
Specifies the frequency multiplication factor. A value of 3 means that the generated clock period is one-third as long as the master clock period.

**-duty_cycle** *percent*
Specifies the duty cycle as a percentage, when frequency multiplication is used. Duty cycle is the high pulse width.

**-invert**
Inverts the generated clock signal (for frequency multiplication or division).

**-edges** *edgeList*
Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be set to 3 to make one full clock cycle of the generated clock waveform.

**-edge_shift {***edgeShiftList***}**
Specifies a list of floating point numbers that represents the amount of shift, in nanoseconds, that the specified edges are to undergo to yield the final generated clock waveform. The number of edge shifts specified must be equal to the number of edges specified. The values can be positive or negative; positive indicates a shift later in time, and negative indicates an earlier shift. For example, 1 indicates that the corresponding edge is to be shifted by one library time unit.

**-combinational**
Includes only the logic for this type of generated clock in the source latency paths where the master clock propagates. The source latency paths do not flow through sequential element clock pins, transparent latch data pins, or source pins of other generated clocks.

**-disable**
Disables the constraint.

**-comment** *textString*

Annotates the object. The tool preserves the comment with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, timing analysis, and place and route.

## Examples

### Example 1: Setting Generated Clock Frequency of Divide by 2

```
create_generated_clock -name {gen_clk} -source
   [get_pins {DCM0.CLK0}] [get_pins {BUFGMUX_inst.O}] -divide_by 2
```

### Example 2: Setting Generated Clock Edges

Sets the edges of the generated clock to be 1, 3, and 5 of the master clock source. If the master clock period is 30 and the master waveform is {24 36}, then the generated clock period becomes 60 with waveform {24 54}.

```
create_generated_clock -name {genclk} -source
   [get_ports {clk_in1}] [get_nets {dut.clk_out2}] -edges {1 3 5}
```

### Example 3: Shifting Generated Clock Edge

This example shifts the derived edges from the previous example by 1 time unit. If the master clock period is 30 and the master waveform is {24 36}, then the generated clock period becomes 60 with waveform {25 55}.

```
create_generated_clock -name {genclk}
   -source [get_ports {clk_in1}] [get_nets {dut.clk_out2}]
   -edges {1 3 5} -edge_shift {1 1 1}
```

### Example 4: Setting Generated Clock from Same Clock Edges

This example shows the generated clock with the same edges as the master clock, where edge 2 is shifted by 0.8 time unit and edge 3 is shifted by -0.4 time unit. If the master clock period is 4 and the master waveform is {0 2}, then the generated clock period becomes 3.6 and the waveform is {0 2.8}.

```
create_generated_clock -name {genclk}
   -source [get_ports {clk_in1}] [get_nets {dut.clk_out2}]
   -edges {1 2 3} -edge_shift {0 0.8 -0.4}
```

### Example 5: Renaming an Automatically Generated Clock

```
create_generated_clock -name {mem_clk} [get_pins {mmcm0.CLKOUT0}]
```

# create_clockgen_group

Specifies a list of clocks to be disconnected from the original source and generated from the clock generator.

Note that clocks can be generated additionally from PLLs/top level IOs, but these clocks will not be controlled by the clock generator. Generate such clocks only to drive peripheral IO devices.

## Syntax

**create_clockgen_group**
*listOfClocks*
**-group** *clkGrp*

*listOfClocks*
List of clocks to be generated.

**-group** *clkgrp*
Name of the clock group to be created using the clock generator.

## Example

If the following clocks are defined in FDC file:

```
create_clock -name clk1 -period 200 [get_ports {clk1}] -waveform
{0 100}
create_clock -name clk2 -period 300 [get_ports {clk2}] -waveform
{0 150}
create_clock -name clk3 -period 400 [get_ports {clk3}] -waveform
{0 200}
```

Add the following command in the FDC file to generate clk2 and clk3 from the HAPS clock generator:

```
create_clockgen_group {clk2 clk3} -group {group_1}
```

# define_haps_fpga

Specifies the board type and FPGA device location for the define_haps_io command.

## Syntax

**define_haps_fpga -type {***hapsBoardType***}**
**-location FB***cdeChainInteger_deviceLocation*

**-type {***hapsBoardType***}**
Specifies the type of HAPS system. Accepted type values are:

- HAPS-70_S12, HAPS-70_S24, and HAPS-70_S48 for HAPS-70 systems
- HAPS-80_S26, HAPS-80_S52, and HAPS-80_S104 for HAPS-80 systems

**-location FB***cdeChainInteger_deviceLocation*
Specifies the default CAPIM address (*cdeChainInteger*) and FPGA device location on the HAPS system. The *cdeChainInteger* argument sets the integer address of the default CAPIM; address values range from 1 through 8. The *deviceLocation* argument is a letter value that specifies the FPGA device location on the specified HAPS board type. Individual FPGAs are identified by the letters A through D.

## Example

The following example identifies FPGA C on CAPIM 4 on the CDE chain for a HAPS-80 S104 system:

```
define_haps_fpga -type {HAPS-80_S104} -location FB4_C
```

# define_haps_io

Maps Xilinx device-pin locations to HAPS board-connector locations for single-FPGA designs. For multi-FPGA (partitioned) designs see board_system_create in TSS with assign_port in PCF.

---

**Note:** HAPS-80D does not support synthesis flow. Hence this constraint is not supported on HAPS-80D.

---

For more information on using this constraint, see Running Pre-Map, on page 474 in the *User Guide*.

## Syntax

**define_haps_io {p:***portname***} -haps_io {***hapsConnectorName***}**
**[-add_clock_deskew PLL|MMCM]** [**-comment** *textString*]

**{p:***portname***}**
The name of a signal on a top-level netlist port.

**-haps_io {***hapsConnectorName***}**
The corresponding HAPS connector name for the top-level netlist port.

**-add_clock_deskew PLL|MMCM**
Optional argument that adds clock deskew for PLL- or MMCM-generated HAPS GCLKs.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves the string with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Description

HAPS pin mapping defines the relationship between device pin and board connector locations for HAPS systems through internal lookup tables. The constraint is included in an FDC file that is read during the pre-map stage. The connector-pin mappings are listed in the HAPS I/O Report generated during pre-map.

## Example

```
define_haps_io {p:ibus[15:0]} -haps_io {A[10:5],[B[9:0]}
```

# define_io_standard

Specifies a standard I/O pad type. Use this in combination with the syn_padtype attribute.

## Syntax

**define_io_standard** [**-disable**] [**p:***portName*]
[**-default_input**|**-default_output**|**-default_bidir**] **-delay_type {input**|**output**|**bidir}**
**syn_pad_type {***IO_standard***}** [*parameter* **{***value***}** [**...**]] [**-comment** *textString*]

**-disable**
Disables the constraint.

**p:***portName*
The name of an individual port to be defined by the -delay_type argument.

**-default_input**|**-default_output**|**-default_bidir**
Identifies the default standard to be used for the input, output, or bidirectional ports.

**-delay_type {input**|**output**|**bidir}**
Identifies the delay type for the named port (p:*portName*) and overrides the default settings.

**syn_pad_type**
The I/O pad type (I/O standard) to be assigned.

*parameter* **{***value***}**
One or more of the parameters defined in the following table. Use these parameters in combination with the syn_padtype attribute.

| Parameter | Function |
|---|---|
| syn_io_slew | The slew rate for single-ended output buffers; values include slow and fast or low and high. |
| syn_io_drive | The output drive strength; numerical values in mA. |
| syn_io_termination | The termination type; typical values are pullup and pulldown. |
| syn_io_dci | Switch for digitally-controlled impedance (DCI). |
| syn_io_dv2 | Switch to use a 2x impedance value. |

**-comment** *textString*

Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through all of the flows.

## Example

```
define_io_standard -default_input syn_pad_type {LVDS_18}
   syn_io_slew {slow} syn_io_drive {8}
   syn_io_termination {pullup} syn_io_dci {DCI} syn_io_dv2 {DV2}

define_io_standard p:driver1 syn_pad_type {LVDS_25}
   -delay_type {input} syn_io_slew {slow} syn_io_drive {12}
   syn_io_termination {pullup} syn_io_dv2 {DV2}
```

# reset_path

Resets the specified paths to single-cycle timing.

## Syntax

**reset_path** [**-setup**]
[**-from {***objectList***}** |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*]
[**-through {***objectList***}** [**-through {***objectList***} ...**] ]
[**-to {***objectList***}** |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*]
[**-disable**]
[**-comment** *commentString*]

**-setup**
Specifies that setup checking (maximum delay) is reset to single-cycle behavior.

**-from**
Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs
- Sequential cell clock pins

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points.

**-rise_from** *riseFromClock*
Specifies to use the rising edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*
Specifies to use the falling edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-through**
Specifies the intermediate points for the timing exception. The -through *objectList* includes:

- Combinational nets
- Hierarchical ports
- Pins on instantiated cells
- Cell instances

By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, reset_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, reset_path applies only if the -from or -to and the -through conditions are satisfied.

**-to**

Specifies the names of objects to use to find path end points. The -to *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs

If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.

**-rise_to** *riseToClock*

Specifies to use the rising edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*

Specifies to use the falling edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-disable**

Disables the constraint.

**-comment** *textString*

Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is

written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

# set_case_analysis

Specifies that a port or pin is at a constant 1 or 0 logic value.

**Note:** Currently, set_case_analysis is only supported on the select input pin for BUFGMUX, BUFGMUX_1, or BUFGMUX_CTRL.

## Syntax

The supported syntax for the set_case_analysis constraint is:

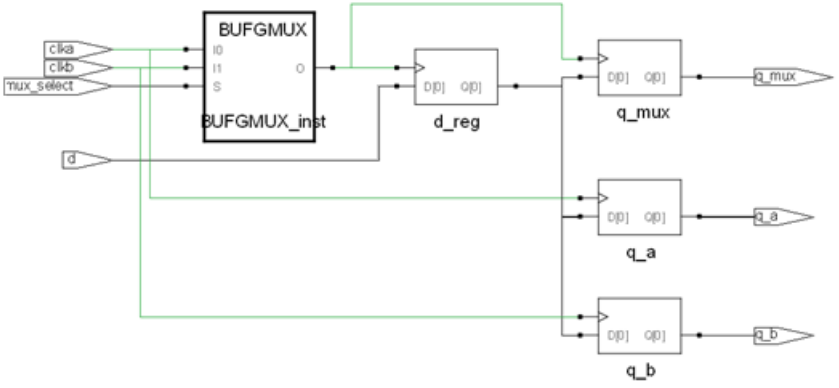
**set_case_analysis 0|1 {***portOrPinList***}**

**0|1**
Specifies that the level of the port or pin is at either a constant logic 1 or a constant logic 0.

**{***portOrPinList***}**
Specifies the port or pin list.

## Examples

The synthesis tool allows multiple clocks to propagate along a single net, either through unate logic or a mux for timing all possible clocks. In the following example, incorrect clock reporting occurs when both clocks are propagated through the same mux.



If both clocks are propagated, the timing report lists the following timing paths:

```
Starting point: d_reg / Q
Ending point: q_mux / D
The start point is clocked by clka [rising] on pin C
The end point is clocked by clkb [rising] on pin C
```

This path is invalid because clka and clkb cannot both exist on the same net. The set_case_analysis constraint lets you select which clock to propagate through the mux.

Specifying this constraint with a query command (get_ports) allows clka to propagate through the mux:

```
set_case_analysis 0 [get_ports {mux_select}]
```

To propagate clkb through the same mux, use the following constraint:

```
set_case_analysis 1 [get_ports {mux_select}]
```

# set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design. Clocks created with create_clock are considered synchronous as long as no set_clock_groups constraints specify otherwise. Paths between asynchronous clocks are not considered for timing analysis. Clock grouping is inclusionary or exclusionary. For example, clk2 and clk3 can each be related to clk1 without being related to each other.

For details about using set_clock_groups, see *Defining Clock Groups, on page 41*.

## Syntax

**set_clock_groups**
**-asynchronous**|**-physically_exclusive**|**-logically_exclusive**
**[-name** *clockGroupname***]**
**-group {***clockList***}** [**-group {***clockList***}** … ]
**-derive**
[**-disable**]
[**-comment** *commentString*]

**-asynchronous**
Specifies that the clock groups are asynchronous to each other (the default assumes that all clock groups are synchronous). Two clocks are asynchronous with respect to each other if they have no phase relationship at all.

**-physically_exclusive**
Specifies that the clock groups are physically exclusive to each other. An example is multiple clocks that are defined on the same source pin. Synthesis accepts this option, but treats it as -asynchronous.

**-logically_exclusive**
Specifies that the clock groups are logically exclusive to each other. An example is multiple clocks that are selected by a multiplexer, but might have coupling with each other in the design. Synthesis accepts this option, but treats it as -asynchronous.

**-name {***clockGroupName***}**
Specifies a unique name for a clock grouping. This option allows you to easily identify specified clock groups, which are exclusive or asynchronous with all other clock groups in the design.

**-group {*clockList*}**

Specifies a space-separated list of clocks in {*clockList*}. Clocks in the same group synchronous with each other, but depending on how the constraint is specified, these clocks are asynchronous to all other clocks in the design, or asynchronous to the clocks specified in other -group arguments.

Do not use commas between clock names in the list.

All clocks start out in the default group, unless they are assigned to another group. When a new clock is created, it is automatically placed in the default group.

You can set the constraint with a single -group argument, and use multiple such constraints to constrain your clocks, or you can use a single constraint with multiple -group arguments. See *Examples of Asynchronous Clocks*, on page 310.

- If you specify only one -group, the clocks in that group are synchronous with each other, but exclusive or asynchronous with all other clocks in the design. You can specify multiple such constraints.
- If you specify -group multiple times in a single constraint, the listed clocks in one group are only asynchronous with the clocks in the other groups specified in the same constraint.

You can include a clock in only one group in a single constraint. To include a clock in multiple groups, use multiple set_clock_groups commands.

**-derive**

Specifies that generated and derived clocks inherit the clock group of the parent clock. By default, a generated clock and its master clock are not in the same group when the exclusive or asynchronous clock groups are defined. The -derive option overrides this behavior and allows generated or derived clocks to inherit the clock group of their parent source clock.

**-disable**

Disables the constraint.

**-comment** *textString*

Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Restrictions

Be aware of the restrictions for the following set_clock_groups options:

### -group Option

Do not insert commas between clock names when you use the -group option, because the tool treats the comma as part of the clock name. This is true for all constraints that contain lists. This means that if you specify the following constraint, the tool generates a warning that it cannot find clk1,:

```
set_clock_groups -asynchronous -group {clk1, clk2}
```

## Examples

The following examples illustrate how to use this constraint.

### Example 1

This set_clock_groups constraint specifies that clk4 is asynchronous to all other clocks in the design.

```
set_clock_groups -asynchronous -group {clk4}
```

### Example 2

This set_clock_groups constraint specifies that clock clk1, clk2, and clk3 are asynchronous to all other clocks in the design. If a new clock called clkx is added to the design, clk1, clk2, and clk3 are asynchronous to it too.

```
set_clock_groups -asynchronous -group {clk1 clk2 clk3}
```

### Example 3

The following set_clock_groups constraint has multiple -group arguments, and specifies that clk1 and clk2 are asynchronous to clk3 and clk4.

```
set_clock_groups -asynchronous -group {clk1 clk2}
   -group {clk3 clk4}
```

### Example 4

The following set_clock_groups constraint specifies that clk1 and clk2 which were synchronous when defined with the create_clock command, are now asynchronous.

```
create_clock [get_ports {c1}] -name clk1 -period 10
create_clock [get_ports {c2}] -name clk2 -period 16
create_clock [get_ports {c3}] -name clk3 -period 5
set_clock_groups -asynchronous -group [get_clocks {clk1}]
   -group [get_clocks {clk2}]
```

The following constructs are equivalent:

```
set_clock_groups -asynchronous -group [get_clocks {clk1}]
```

```
set_clock_groups -asynchronous -group {clk1}
```

### Example 5

The following constraint specifies that test|clkout0_derived_clock_CLKIN1 and test|clkout1_derived_clock_CLKIN1 are asynchronous to all other clocks in the design:

```
set_clock_groups -asynchronous -group [get_clocks {*clkout*}]
```

### Example 6

This example defines the clock on the u1.clkout0 net is asynchronous to all other clocks in the design:

```
set_clock_groups -asynchronous -group [get_clocks -of_objects
{n:u1.clkout0}]
```

## Examples of Asynchronous Clocks

### Example 1: Multiple -group Arguments for Asynchronous Clock Definition

This method uses multiple -group arguments in one constraint:

```
set_clock_groups -asynchronous -group {clk1 clk2} -group {clk3
bclk4} -group {clk5 cclk6}
```

With this constraint, members of the same group are synchronous, but relationships between clocks from different groups defined in this constraint are asynchronous. This has the following implications:

- clk1 and clk2 are synchronous to each other, but asynchronous to clocks in all other groups defined in this constraint
- clk3 and clk4 are synchronous to each other, but asynchronous to clocks in all other groups defined in this constraint

- clk5 and clk6 are synchronous to each other, but asynchronous to clocks in all other groups defined in this constraint

### Example 2: Single -group Argument for Asynchronous Clock Definition

Asynchronous clocks defined with a single -group argument in a constraint are asynchronous to all other clocks in the design. You can specify multiple such constraints. In this example, all six clocks are asynchronous, because each individual constraint makes that clock asynchronous to all others.

```
set_clock_groups -asynchronous -group {clk1}
set_clock_groups -asynchronous -group {clk2}
set_clock_groups -asynchronous -group {clk3}
set_clock_groups -asynchronous -group {clk4}
set_clock_groups -asynchronous -group {clk5}
set_clock_groups -asynchronous -group {clk6}
```

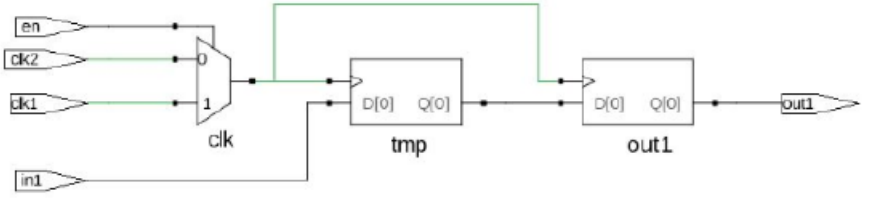
### Example 3: Propagating Exception Constraints

By default, exception constraints are not propagated through MMCM/PLL or to generated/derived clocks. The following example shows how to propagate exception constraints on generated/derived clocks:

```
set_clock_groups -asynchronous -group [get_clocks -include_generated_clocks GCLK2] -group [get_clocks -include_generated_clocks GCLK1]
```

## Examples of Defining Clocks for Clock Muxes

The definition of clocks that are to be muxed together varies slightly, depending on whether the clocks have the same frequency or not. The following procedures use this example as an illustration:




### Defining Muxed Clocks with Different Frequencies

If the clocks are asynchronous, separate clock paths must be defined, as described below.

1. Define the clocks with create_clock constraints.

   For the example, two clocks are defined:

   ```
   create_clock -name {clk1} [get_nets {clk1}] -period 10.0
       -waveform {0 5.0}
   create_clock -name {clk2} [get_nets {clk2}] -period 10.0 -waveform {0 5.0}
   ```

2. Use multiple set_clock_groups constraints to mark them as asynchronous to each other:

   ```
   set_clock_groups -derive -asynchronous -name {default_clkgroup_0} -group
       [get_clocks {clk1}]
   set_clock_groups -derive -asynchronous -name {default_clkgroup_1} -group
       [get_clocks {clk2}]
   ```

3. Check the timing report.

   For the example, the tool reports two separate clock paths, one for each clock.

## Defining Muxed Clocks with the Same Frequency

If the clocks have the same phase and frequency, follow this procedure to define the clocks.

1. Define the clock at the net connected to the output pin of the mux.

   For example:

   ```
   create_clock -name {clk} [get_nets {clk}] -period 10.0
       -waveform {0 5.0}
   ```

2. Define the mux output clock as asynchronous to all other clocks, using a set_clock_groups constraint:

   ```
   set_clock_groups -derive -asynchronous -name {default_clkgroup_2} -group
       [get_clocks {clk}]
   ```

3. Check the timing report.

   In this case, there should be a single clock path, instead of separate paths.

# set_clock_latency

Specifies clock network latency. The constraint accepts both clock objects and clock aliases.

## Syntax

**set_clock_latency**
**-source**
[**-clock {***clockList***}**]
*delayValue*
**{***objectList***}**
[**-disable**]

**-source**
Indicates that the specified delay is applied to the clock source latency.

**-clock** *clockList*
Indicates that the specified delay is applied with respect to the specified clocks. By default, the specified delay is applied to all specified objects.

*delayValue*
Specifies the clock latency value.

*objectList*
Specifies the input ports for which clock latency is to be set and references either input ports with defined clocks or clock aliases defined on the input ports. When more than one clock is defined for an input port, the -clock option can be used to apply different latency values to each alias.

**-disable**
Disables the constraint.

# set_clock_uncertainty

Specifies the uncertainty (skew) of the specified clock networks.

## Syntax

**set_clock_uncertainty**
**{***objectList***}**
**-from** *fromClock* |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*
**-to** *toClock* |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*
*value*

*objectList*
Specifies the clocks for simple uncertainty. The uncertainty is applied to the capturing latches clocked by one of the specified clocks. Specify either this argument or a clock pair with the -from/-rise_from/-fall_from and -to/-rise_to/-fall_to options; you cannot specify both an object list and a clock pair.

**-from** *fromClock*
Specifies the source clocks for inter-clock uncertainty. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-rise_from** *riseFromClock*
Specifies that the uncertainty applies only to the rising edge of the source clock. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*
Specifies that the uncertainty applies only to the falling edge of the source clock. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-to** *toClock*
Specifies the destination clocks for inter-clock uncertainty. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-rise_to** *riseToClock*
: Specifies that the uncertainty applies only to the rising edge of the destination clock. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*
: Specifies that the uncertainty applies only to the falling edge of the destination clock. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

*value*
: Specifies a floating-point number that indicates the uncertainty value. Only positive clock uncertainty numbers are acceptable.

## Examples

Refer to the following examples.

### Example 1

All paths to registers clocked by clk are specified with setup uncertainty of 0.4 in the following example:

```
set_clock_uncertainty 0.4 -setup [get_clocks clk]
```

### Example 2

For this example, interclock uncertainties are specified between clock clk and clk2:

```
set_clock_uncertainty -from [get_clocks clk] -to
   [get_clocks clk2] 0.2

set_clock_uncertainty -from [get_clocks clk2] -to
   [get_clocks clk] 0.1
```

### Example 3

For this example, interclock uncertainties are specified between clock clk and clk2 with specific edges:

```
set_clock_uncertainty -rise_from [get_clocks clk2] -to
   [get_clocks clk] 0.5
```

```
set_clock_uncertainty -rise_from [get_clocks clk2] -rise_to
   [get_clocks clk] 0.1

set_clock_uncertainty -from [get_clocks clk2] -fall_to
   [get_clocks clk] 0.1
```

# set_datapathonly_delay

Specifies a point-to-point path delay that excludes any applicable clock delays at the start and end points. The constraint includes clock to out (Tco) and setup time (Tsu) as part of the data path delay. Using this constraint ignores any clock skew or phase information, and constrains, analyzes, and forward-annotates just the data path between the specified points.

The set_datapathonly_delay constraint appears in the XDC file with the set_max-_delay -datapath_only option. The original .fdc command is forward-annotated as a comment. For this example, the -from and -to options are used to specify the pins for the start and end points of the path delay.

```
#set_datapathonly_delay -from {p:din[*]} -to {i:int[*]} {4}

set_max_delay -datapathonly_delay -from [get_ports {d_in[*]}]
   -to [get_cells {int[*]}] {4}
```

## Syntax

**set_datapathonly_delay** *option* **{***value***}**

In the above syntax, *option* is one or more of the following:

**-comment** *textString*
**-disable**
**-from** *fromClock* |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*
**-to** *toClock* |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*
**-through {***objectList***}** [**-through {***objectList***} ...**]

*value*
Specifies a floating-point number that indicates the delay value

**-from {***port*|*inst*|*clock***}**
Specifies the start point for the path delay.

**-rise_from** *riseFromClock*
Specifies that the start delay applies only to the rising edge of the source clock. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*
Specifies that the start delay applies only to the falling edge of the source clock. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-to {***port*|*inst*|*clock***}**
Specifies the end point for the path delay.

**-through {***objectList***}** [**-through {***objectList***} ...**] ]
Specifies the intermediate points for the timing exception. The -through *objectList* includes:

- Combinational nets
- Hierarchical ports
- Pins on instantiated cells
- Cell instances

By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, set_false_path applies only if the -from or -to and the -through conditions are satisfied.

**-rise_to** *riseToClock*
Specifies that the end delay applies only to the rising edge of the source clock. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*
Specifies that the end delay applies only to the falling edge of the source clock. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-disable**
Disables the constraint.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

In the above options, the variable *clock* is a clock alias.

## References

- Time Budgeting, on page 146 in the *Reference Manual*

- Generating FPGAs, on page 416 in the *User Guide*

# set_false_path

Removes timing constraints from designated paths.

## Syntax

**set_false_path**
[**-setup**]
[**-from {***objectList***}** |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*]
[**-through {***objectList***}** [**-through {***objectList***} ...**] ]
[**-to {***objectList***}** |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*]
[**-disable**]
[**-comment** *commentString*]

**-setup**
Specifies that setup checking (maximum delay) is reset to single-cycle behavior.

**-from**
Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs
- Sequential cell clock pins
- Sequential cell data output pins

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points.

**-rise_from** *riseFromClock*
Specifies to use the rising edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*
Specifies to use the falling edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-through**

Specifies the intermediate points for the timing exception. The -through *objectList* includes:

- Combinational nets
- Hierarchical ports
- Pins on instantiated cells
- Cell instances

By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, reset_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, reset_path applies only if the -from or -to and the -through conditions are satisfied.

**-to**

Specifies the names of objects to use to find path end points. The -to *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs

If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.

**-rise_to** *riseToClock*

Specifies to use the rising edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*

Specifies to use the falling edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-disable**

Disables the constraint.

**-comment** *textString*

Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

Refer to the following examples.

### Example 1

All the paths from the sequential cell output pins and clock pins in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) are set as false paths.

```
set_false_path -from [get_pins {gen_sub\[*\]\.u_sub.out*.*}]
```

---

**Note:** Only sequential cell clock pins are pins that can be used as valid start points for a timing path. Note that hierarchical module pins are not valid as starting points for a timing path.

---

### Example 2

All the paths from the sequential cells in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) are set as false paths.

```
set_false_path -from [get_cells {gen_sub\[*\]\.u_sub.out*}]
```

### Example 3

All paths from top-level input ports with names in* are set as false paths.

```
set_false_path -from [get_ports {in*}]
```

**Note:** Only top-level ports are valid port based start points for timing paths. Do not use the get_ports command to reference hierarchical module pins.

### Example 4

All paths with end points clocked by clock clka are set as false paths.

```
set_false_path -to [get_clocks {clka}]
```

### Example 5

By default, exception constraints are not propagated through MMCM/PLL or to generated/derived clocks. The following example shows how to propogate exception constraints on generated/derived clocks:

```
set_false_path -from [get_clocks -include_generated_clocks GCLK2] -to
[get_clocks -include_generated_clocks GCLK1]
```

# set_input_delay

Sets input delay on pins or input ports relative to a clock signal.

## Syntax

**set_input_delay**
[**-clock** *clockName* [**-clock_fall**]]
[**-rise**|**-fall**]
[**-min**|**-max**]
[**-add_delay**]
*delayValue*
**{***portPinList***}**
[**-disable**]
[**-comment** *commentString*]

**-clock** *clockName*
Specifies the clock to which the specified delay is related. If -clock_fall is used, -clock *clockName* must be specified. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.

**-clock_fall**
Specifies that the delay is relative to the falling edge of the clock. The default is the rising edge.

**-rise**
Specifies that *delayValue* refers to a rising transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. No differentiation is made between the rising and falling edges for the data transition arcs on the specified ports, and the worst-case path delay is used instead. The -rise option is preserved and forward annotated to the place-and-route tool.

**-fall**
Specifies that *delayValue* refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. No differentiation is made between the rising and falling edges for the data transition arcs on the specified ports, and the worst-case path delay is used instead. The -fall option is preserved and forward annotated to the place-and-route tool.

**-min**

Specifies that *delayValue* refers to the shortest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal. Hold time violations are not optimized and only -min delay values are reported in the log file. The -min delay values are forward annotated to the place-and-route tool.

**-max**

Specifies that *delayValue* refers to the longest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal. The -max delay values are reported in the log file and are forward annotated to the place-and-route tool.

**-add_delay**

Indicates if delay information is to be added to the existing input delay or if the existing delay is to be overwritten. The -add_delay option allows the capture of information about multiple paths leading to an input port that are relative to different clocks or clock edges.

*delayValue*

Specifies the path delay. This value is the amount of time that the signal is available after a clock edge and represents a combinational path delay from the clock pin of a register.

*portPinList*

Specifies a list of input port names in the current design to which *delayValue* is assigned. When more than one object is specified, enclose the objects in quotes ("") or in braces ({}).

**-disable**

Disables the constraint.

**-comment** *textString*

Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

Refer to the following examples.

### Example 1

This example sets an input delay of 1.0 relative to the rising edge of clk.

```
set_input_delay 1.00 -clock clk [get_ports {din1 din2}]
```

### Example 2

The following example sets an input delay of 1.0 relative to the rising edge of clk for all inputs in the design.

```
set_input_delay 1.00 -clock clk [all_inputs]
```

### Example 3

In this scenario, there are two paths to the input port din1. The input delay for the first path is relative to the rising edge of clk. For the second path, the input delay is relative to the falling edge of clk. The -add_delay option indicates that the new input delay information does not cause old information to be removed.

```
set_input_delay 1.00 -clock clk [get_ports {din1}]

set_input_delay 2.00 -clock clk [get_ports {din1}] -add_delay
   -clock_fall
```

# set_max_delay

Specifies a maximum delay target for paths in the current design.

## Syntax

**set_max_delay**
[**-from {***objectList***}** |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*]
[**-through {***objectList***}** [**-through {***objectList***} ...**] ]
[**-to {***objectList***}** |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*]
*delayValue*
[**-disable**]
[**-comment** *commentString*]

**-from {***objectList***}**
Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs
- Sequential cell clock pins

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. All paths from these start points to the end points in the -from *objectList* are constrained to *delayValue*. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, enclose the objects in quotation marks ("") or braces ({}).

**-rise_from** *riseFromClock*
Specifies to use the rising edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*
Specifies to use the falling edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-through**

Specifies the intermediate points for the timing exception. The -through *objectList* includes:

- Combinational nets
- Hierarchical ports
- Pins on instantiated cells
- Cell instances

By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. The max delay value applies only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_max_delay applies to the paths that pass through a member of each *objectList*. When using the -through option in combination with the -from or -to options, set_max_delay applies only if the -from or -to and the -through conditions are satisfied.

**-to**

Specifies the names of objects to use to find path end points. The -to *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs

If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. All paths to the end points in the -to *objectList* are constrained to *delayValue*. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. When including more than one object, enclose the objects in quotation marks ("") or braces ({}).

**-rise_to** *riseToClock*

Specifies to use the rising edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*
: Specifies to use the falling edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

*delayValue*
: Specifies the value of the desired maximum delay for paths between start and end points. If a path start point is on a sequential device, clock skew is included in the computed delay. If a path start point has an input delay specified, that delay value is added to the path delay. If a path end point is on a sequential device, clock skew and library setup time are included in the computed delay. If the end point has an output delay specified, that delay is added into the path delay.

**-disable**
: Disables the constraint.

**-comment** *textString*
: Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

This example shows how to specify that all paths from cell temp1 to cell temp2 must be less than 4.0 units.

```
set_max_delay -from [get_cells {temp1}] -to [get_cells {temp2}] 4
```

# set_min_delay

Specifies a minimum delay target for paths in the current design.

## Syntax

**set_min_delay**
[**-from {***objectList***}** |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*]
[-**through {***objectList***}** [**-through {***objectList***} ...**] ]
[**-to {***objectList***}** |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*]
*delayValue*
[**-disable**]
[**-comment** *commentString*]

**-from {***objectList***}**
Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs
- Sequential cell clock pins

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. All paths from these start points to the end points in the -from *objectList* are constrained to *delayValue*. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, enclose the objects in quotation marks ("") or braces ({}).

**-rise_from** *riseFromClock*
Specifies to use the rising edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*
Specifies to use the falling edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-through**

Specifies the intermediate points for the timing exception. The -through *objectList* includes:

- Combinational nets
- Hierarchical ports
- Pins on instantiated cells
- Cell instances

By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. The minimum delay value applies only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_min_delay applies to the paths that pass through a member of each *objectList*. When using the -through option in combination with the -from or -to options, set_min_delay applies only if the -from or -to and the -through conditions are satisfied.

**-to**

Specifies the names of objects to use to find path end points. The -to *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs

If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. All paths to the end points in the -to *objectList* are constrained to *delayValue*. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. When including more than one object, enclose the objects in quotation marks ("") or braces ({}).

**-rise_to** *riseToClock*

Specifies to use the rising edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*
: Specifies to use the falling edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

*delayValue*
: Specifies the value of the desired minimum delay for paths between start and end points. If a path start point is on a sequential device, clock skew is included in the computed delay. If a path start point has an input delay specified, that delay value is added to the path delay. If a path end point is on a sequential device, clock skew and library setup time are included in the computed delay. If the end point has an output delay specified, that delay is added into the path delay.

**-disable**
: Disables the constraint.

**-comment** *textString*
: Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

This example shows how to specify that all paths from cell temp1 to cell temp2 must be greater than 4.0 units.

```
set_min_delay -from [get_cells {temp1}] -to [get_cells {temp2}] 4
```

# set_multicycle_path

Modifies the single-cycle timing relationship of a constrained path.

## Syntax

The supported syntax for the set_multicycle_path constraint is:

**set_multicycle_path**
[**-setup** |**-hold**]
[**-start** |**-end**]
[**-from {***objectList***}** |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*]
[**-through {***objectList***}** [**-through {***objectList***} ...**] ]
[**-to {***objectList***}** |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*]
*pathMultiplier*
[**-disable**]
[**-comment** *commentString*]

**-setup** |**-hold**

The option -setup specifies the pathMultiplier to be used for the setup (maximum delay) calculations.

The option -hold enables you to over-ride the default hold multiplier—{pathMultiplier – 1}— that is forward annotated to the vendor constraint file. If you use this option, you must specify the hold value for each of the defined multicycle constraints.

If you do not provide -setup or -hold, the pathMultiplier is used for setup.

**-start|-end**

Indicates if the multi-cycle information is relative to the period of either the start clock or the end clock. These options are only needed for multi-frequency designs; otherwise start and end are equivalent. The start clock is the clock source related to the register or primary input at the path start point. The end clock is the clock source related to the register or primary output at the path endpoint. The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with -end moves the relation forward one cycle of the end clock. A setup multiplier of 2 with -start moves the relation back one cycle of the start clock. A hold multiplier of 1 with -start moves the relation forward one cycle of the start clock. A hold multiplier of 1 with -end moves the relation back one cycle of the end clock.

**-from {***objectList***}**

Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs
- Sequential cell clock pins
- Sequential cell data output pin

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. All paths from these start points to the end points in the -from *objectList* are constrained to *delayValue*. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, enclose the objects in quotation marks ("") or braces ({}).

**-rise_from** *riseFromClock*

Specifies to use the rising edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-fall_from** *fallFromClock*

Specifies to use the falling edge of the source clock to find path start points. Use only one of the -from, -rise_from, and -fall_from options and specify a destination clock with one of the -to, -rise_to, and -fall_to options.

**-through {***objectList***}**

Specifies the intermediate points for the timing exception. The -through *objectList* includes:

- Combinational nets
- Hierarchical ports
- Pins on instantiated cells
- Cell instances

The multi-cycle values apply only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, enclose the objects either in double quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_multicycle_delay applies to the paths that pass through a member of each *objectList*. If

the -through option is used in combination with the -from or -to options, the multi-cycle values apply only if the -from or -to conditions and the -through conditions are satisfied.

**-to {***objectList***}**
Specifies the names of objects to use to find path end points. The -to *objectList* includes:

- Clocks
- Registers
- Top-level input or bidirectional ports
- Black box outputs

If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. When including more than one object, enclose the objects in quotation marks ("") or braces ({}).

**-rise_to** *riseToClock*
Specifies to use the rising edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

**-fall_to** *fallToClock*
Specifies to use the falling edge of the source clock to find path end points. Use only one of the -to, -rise_to, and -fall_to options and specify a source clock with one of the -from, -rise_from, and -fall_from options.

*pathMultiplier*
Specifies the number of cycles that the data path must have, relative to the start point or end point clock, before data is required at the end point.

**-disable**
Disables the constraint.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

Refer to the following examples.

### Example 1

All the paths from the sequential cell output pins and clock pins in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -from [get_pins{gen_sub\[*\]\.u_sub.out*.*}] 2
```

**Note:** Only sequential cell clock pins are pins that can be valid start points for a timing path. Note that hierarchical module pins cannot be used as starting points for a timing path.

### Example 2

All the paths from the sequential cells in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) support the timing cycle set to 2.

```
set_multicycle_path -from [get_cells {gen_sub\[*\]\.u_sub.out*}] 2
```

### Example 3

All paths from top-level input ports with names in* provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -from [get_ports {in*}] 2
```

**Note:** Only top-level ports are valid port based start points for timing paths. Do not use the get_ports command to reference hierarchical module pins.

### Example 4

All paths with end points clocked by clock clka provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -to [get_clocks {clka}] 2
```

# set_output_delay

Sets output delay on pins or output ports relative to a clock signal.

## Syntax

The supported syntax for the set_output_delay constraint is:

**set_output_delay**
[**-clock** *clockName* [**-clock_fall**]]
[**-rise**|[**-fall**]
[**-min**|**-max**]
[**-add_delay**]
*delayValue*
**{***portPinList***}**
[**-disable**]
[**-comment** *commentString*]

**-clock** *clockName*
Specifies the clock to which the specified delay is related. If -clock_fall is used, -clock *clockName* must be specified. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.

**-clock_fall**
Specifies that the delay is relative to the falling edge of the clock. The default is the rising edge.

**-rise**
Specifies that *delayValue* refers to a rising transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. No differentiation is made between the rising and falling edges for the data transition arcs on the specified ports, and the worst-case path delay is used instead. The -rise option is preserved and forward annotated to the place-and-route tool.

**-fall**
Specifies that *delayValue* refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. No differentiation is made

between the rising and falling edges for the data transition arcs on the specified ports, and the worst-case path delay is used instead. The -fall option is preserved and forward annotated to the place-and-route tool.

**-min**

Specifies that *delayValue* refers to the shortest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal. Hold time violations are not optimized and only -min delay values are reported in the log file. The -min delay values are forward annotated to the place-and-route tool.

**-max**

Specifies that *delayValue* refers to the longest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal. The -max delay values are reported in the log file and are forward annotated to the place-and-route tool.

**-add_delay**

Indicates if delay information is to be added to the existing output delay or if the existing delay is to be overwritten. The -add_delay option allows the capture of information about multiple paths leading to an output port that are relative to different clocks or clock edges.

*delayValue*

Specifies the path delay. This value is the amount of time that the signal is required before a clock edge. For maximum output delay, this value usually represents a combinational path delay to a register plus the library setup time of that register. For minimum output delay, this value is usually the shortest path delay to a register minus the library hold time.

*portPinList*

Specifies a list of output port names in the current design to which *delayValue* is assigned. When specifying more than one object, enclose the objects in quotes ("") or in braces ({}).

**-disable**

Disables the constraint.

**-comment** *textString*

Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

## Examples

Refer to the following examples.

### Example 1

This example sets an output delay of 1.00 relative to the rising edge of clk for all the output ports in the design.

```
set_output_delay 1.00 -clock clk [all_outputs]
```

### Example 2

The following example sets an output delay of 2.00 relative to the falling edge of clk for the output port dout1. The -add_delay option indicates that this delay value is to be added to any existing output delays defined on this port.

```
set_output_delay 2.0 -clock clk [get_ports {dout1}] -add_delay
   -clock_fall
```

# set_reg_input_delay

Speeds up paths feeding a register by a given number of nanoseconds.

## Syntax

**set_reg_input_delay {***registerName***}**
[**-route** *ns*]
[**-disable**]
[**-comment** *textString*]

*registerName*
Specifies a single bit, an entire bus, or a slice of a bus.

**-route**
Tightens constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register.

**-disable**
Disables the constraint.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

# set_reg_output_delay

Speeds up paths coming from a register by a given number of nanoseconds.

## Syntax

**set_reg_output_delay {***registerName***}**
[**-route** *ns*]
[**-disable**]
[**-comment** *textString*]

*registerName*
Specifies a single bit, an entire bus, or a slice of a bus.

**-route** *ns*
Tightens constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register.

**-disable**
Disables the constraint.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

SYNOPSYS®
Silicon to Software™

CHAPTER 7

# Netlist Edit Command Reference

This chapter describes the netlist editing commands which are applied from a Tcl file directly to the named `srs` netlist file read by the run pre_map task. The commands are listed in the following table:

| | | |
|---|---|---|
| connect_net | disconnect_connector | set_property |
| copy_view | disconnect_net | stub_inst |
| create_cell | get_net | stub_view |
| create_cgm | insert_buffer | swap_instance |
| create_instance | load_library | tie_net |
| create_net | remove_buffer | tie_pin |
| create_port | remove_instance | |
| create_process_hierarchy | remove_net | |
| define_current_view | remove_port | |

# connect_net

Connects pins and ports from *objectList* to the net specified in *netName*. The command can be used to connect objects across any hierarchy with the -hier option. Any additional ports for sub-hierarchies can be created as needed. Objects can be single bit or multiple bits, but they must all be of the same size. Multiple-bit objects must be enclosed in curly brackets. With multi-bits objects, the connection is bit-to-bit, following the declarative order. For example, connect_net foo[1:0] mem1.addr[6:7] connects pin mem1.addr[6] to net foo[1] and connects pin mem1.addr[7] to net foo[0].

When using the -hier option to connect objects across hierarchies, the connection cannot be made if there is an intermediate hierarchy marked hard or fixed in the path.

## Syntax

**connect_net** *netName* [**-hier**] *objectList*

*netName*
The name of the net to be connected.

*objectList*
The list of pins and/or ports to be connected to the specified net.

**-hier**
Enables objects to be connected across hierarchies.

## Examples

```
connect_net fast_cpm_clock clk_in u2.clk_in

connect_net {busA[15:0]} {dataA[15:0]}
```

# copy_view

Duplicates a view of the current view with a name of *newView* (the copy_view command only copies hierarchical instances and does not copy Synplify primitives). When the optional -bbox argument is included, *newView* is copied as a black box with only the interconnect in place and without any internal logic. The copy_view command is commonly used to "uniquify" shared views.

## Syntax

**copy_view** *currentView newView* [**-bbox**]

*currentView*
Name of the view to be duplicated.

*newView*
Name of new duplicated view.

**-bbox**
Treats new duplicated view as a black box.

## Examples

When devices U1 and U2 use the same view, the following sequence can be used to create a unique view for U2:

```
copy_view v:origview v:origview1
swap_instance U2 v:origview1
```

# create_cell

Creates an empty view. Add content to the view using the create_port, create_instance, and create_net commands. The *cellName* argument overwrites the view name contained in the netlist file to resolve conflicts with view names. An error message is generated when the name of the view (from *cellName*) conflicts with the name of an existing view.

## Syntax

**create_cell** *cellName*

*cellName*
The name of the view you want to create.

# create_cgm

Inserts an instance of a clock generator module (CGM) in the specified top-level netlist view and connects the module ports to the corresponding nets; an existing, driven clock net is disconnected.

## Syntax

**create_cgm -inst** *cgmInstanceName* **-view** *cgmViewName* [**-warnonly**]

**-inst**
Specifies the instance name of the clock-generator module to be inserted.

**-view**
Specifies the view on the top-level netlist where the clock generator instance is to be inserted.

**-warnonly**
An optional argument that, when present, causes a warning to be generated when the ports on the module do not match the corresponding nets (a port-net mismatch normally results in an error).

## Examples

```
create_cgm -inst u_cgm1 -view clkgen -warnonly
```

**-disable**
Disables the constraint.

**-comment** *textString*
Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

# create_instance

Creates one or more instances in the current view. Hierarchical names allow instance creation below the current level. In the command line, *instanceList* specifies a list of instance names separated by spaces, and *referenceName* specifies the type of view. The view type is either library.cell for a library primitive, or simply view for a netlist created from an `srs` file with the create_cell command. An error message is generated when an instance name (from *instanceList*) conflicts with the name of an existing instance or when *referenceName* does not exist.

## Syntax

**create_instance** *instanceList referenceName*

*instanceList*
A list of instance names separated by spaces.

*referenceName*
The type of view

## Examples

Simple example:

```
create_instance {inst_A nle2_add_select} {FD}
```

The following example illustrates creating an instance of a Xilinx FD primitive from a technology library:

```
load_library $LIB/xilinx/xilinx.syncreate_instance {i:ff_inst}
{FD}
create_port {p:out1} -direction out
create_port {p:in1} -direction in
create_net {n:in1} {n:out1}
connect_net  {n:clk} {t:ff_inst.C}
connect_net {n:in1} {p:in1} {t:ff_inst.D}
connect_net {n:out1} {p:out1} {t:ff_inst.Q}
```

# create_net

Creates new nets in the current view. Multiple net entries in the *netList* are separated by spaces, and both single- and multiple-bit nets can be specified (multiple-bit nets must be enclosed in curly brackets). Hierarchical names allow the creation of nets below the current hierarchy level. An error message is generated when a net name (from *netList*) conflicts with the name of an existing net.

## Syntax

**create_net** *netList*

## Arguments and Options

*netList*
A space-separated list of nets to be added to the netlist.

## Examples

```
create_net fast_cpm_clock

create_net {mem1.addr[7:0]}
```

# create_port

Creates ports of the specified direction in the current view. Multiple port entries in the *portList* are separated by spaces, and both single- and multiple-bit ports can be specified (multiple-bit ports must be enclosed in curly brackets). When no direction is specified, input ports are created. Note that an error message is generated when a port name (from *portList*) conflicts with the name of an existing port.

## Syntax

**create_port** *portList* [**-direction** {**in**|**out**|**inout**}]

*portList*
A space-separated list of ports to be added to the netlist.

**-direction** {**in**|**out**|**inout**}
The direction of the port to be created.

## Examples

```
create_port fast_cpm_clock -direction out

create_port {dataA[7:0]} {dataB[3:0]} -direction inout
```

# create_process_hierarchy

Creates an additional level of hierarchy based on the always (Verilog) or process (VHDL) blocks in the source RTL design file.

## Syntax

**create_process_hierarchy**

# define_current_view

Sets the current view to *viewName*. Because a design can be incrementally uniquified during editing operations, the actual machine-generated view name for a multiply-instantiated module may not be known. When specifying *viewName*, make sure that the entry is correct (no indication is given for a misspelled or non-existent view name).

## Syntax

**define_current_view v:***viewNameviewName*

Name of the view to be set.

## Examples

```
define_current_view v:scenic2
```

# disconnect_connector

Disconnects pins and ports in *objectList* from the nets where they are connected. Objects can be single bit or multiple bits (multiple-bit nets must be enclosed in curly brackets). The command can be used to disconnect objects that are in different hierarchies. An error is generated if the port or pin does not exist or if the port or pin is already disconnected from net.

## Syntax

**disconnect_connector** *objectList*

*objectList*
A list of the ports or pins to be disconnected.

## Examples

The following example disconnects pins mem2.addr[1:0] and mem1.addr[7:0] from their connected nets.

```
disconnect_connector {mem2.addr[1:0]} {mem1.addr[7:0]}
```

# disconnect_net

Disconnects pins and ports in *objectList* from the net specified by *netName*. The command can be used to disconnect objects that are in different hierarchies. Objects can be single bit or multiple bits, but they must all be of the same size. With multi-bits objects, the disconnection is bit-to-bit, following the declarative order. For example, disconnect_net foo[1:0] mem1.addr[6:7] disconnects pin mem1.addr[6] from net foo[1] and disconnects pin mem1.addr[7] from net foo[0].

## Syntax

**disconnect_net** *netName objectList* **| -all**

*netName*
The name of the net.

*objectList*
The list of individual pins and ports to be disconnected from *netName*.

**-all**
Disconnects all pins and ports from *netName*.

## Examples

```
disconnect_net {busA[15:0]} {dataA[15:0]}
```

# get_net

Returns the net name associated with the specified connector. The net must be a single-bit net, and the connector must be an instance pin or a netlist bit port. An error message is generated if the connector is not connected to any net.

## Syntax

**get_net** *connectorName*

*connectorName*
Specifies the connector name with the desired net.

## Example

The following command assigns the net name of pin J1 of instance inst1 to the variable net.

```
set net [get_net t:inst1.J1]
```

# insert_buffer

Inserts buffers on the ports and pins specified in *objectList*. When the -name option is included, *objectlist* is limited to a single port/pin or bus. The buffer type is specified by the *bufferType* argument. A buffer is defined as a view with either a single input or with one or more outputs that function as either input or !input. The -inverter_pair option causes two inverters to be inserted in series.

## Syntax

**insert_buffer** [**-name** *instanceName*] [**-inverter_pair**] *objectList bufferType*

**-name** *instanceName*
The desired instance name of the inserted buffer. Indices are optional and, when included, the indexed name must match the width of the object.

*objectList*
List of the ports and pins to be buffered. When the -name option is included, *objectlist* is limited to a single port/pin bit or bus.

*bufferType*
The type of buffer to be inserted.

**-inverter_pair**
Use two inverters to buffer the port/pin.

## Examples

```
insert_buffer -name MYinv01 -inverter_pair {ux.ub.d[4]} INV

insert_buffer ua.q[7] BUF
```

# load_library

Netlist editing command that loads a primitive library.

## Syntax

**load_library** *primitiveLibrary*

*primitiveLibrary*
The location of the primitive library. For Altera primitives, the library is located in *installDirectory*/lib/altera/altera.syn, and for Xilinx primitives, the library is located in *installDirectory*/lib/xilinx/xilinx.syn.

## Description

The load_library command loads a primitive library for any netlist-editing commands that require the addition or replacement of a vendor primitive within the netlist.

## Examples

```
load_library $LIB/xilinx/xilinx.syn
```

```
load_library $LIB/altera/altera.syn
```

# remove_buffer

Deletes the buffers specified in *instanceList*. A buffer is defined by the insert_buffer command. To remove an inverter pair, the command must list both instances.

## Syntax

**remove_buffer** *instanceList*

*instanceList*
A list of the buffers to be deleted.

# remove_instance

Deletes the specified instances from the current view. Hierarchical names allow deletion of instances below the current level. An error is generated when an instance name (from *instanceList*) does not exist.

## Syntax

**remove_instance** *instanceList* | **-all**

*instanceList*
List of the instances, separated by spaces, to be deleted. Instances can be single bit or multiple bits (multiple-bit instances must be enclosed in curly brackets).

**-all**
Delete all instances from the current view.

# remove_net

Deletes the specified nets from the current view. Hierarchical names allow deletion below the current level. An error is generated when a net name (from *netList*) does not exist.

## Syntax

**remove_net** *netList* | **-all**

*netList*
List of the nets to be deleted. Nets can be single bit or multiple bits (multiple-bit nets must be enclosed in curly brackets)

**-all**
Delete all nets from the current view.

## Examples

```
remove_net {busA[15:0]}
```

# remove_port

Deletes the specified ports from the current view. An error is generated when a port name (from *portList*) does not exist.

## Syntax

**remove_port** *portList* | **-all**

*portList*
Lists the ports to be deleted. Multiple port entries in the *portList* are separated by spaces, and both single- and multiple-bit ports can be specified (multiple-bit ports must be enclosed in curly brackets).

**-all**
Deletes all ports from the current view.

## Examples

```
remove_port fast_cpm_clock {dataA[7:0]}
```

# set_property

Sets properties on objects during netlist editing.

## Syntax

**set_property -type** {**integer**|**string**} *object property value*

**-type** {**integer**|**string**}
The property type; the default is string.

*object*
The name of the object. Object names are prefixed with a letter to identify the object type (v: view, i: instance, p: port, t: pin, n: net). If the object prefix is omitted, instance is assumed.

*property*
The name of the property assigned to the object.

*value*
The value of the assigned property.

## Examples

```
set_property -type string p:portA direction inout
```

# stub_inst

Removes instances and related logic from a design.

## Syntax

**stub_inst** {*instanceName*} **-tie** {*0* | *1*}

*instanceName*
Name of the instance to be removed from the design. To remove more than one instance, separate the names using space.

**-tie** {*0* | *1*}
The constant value with which the module pins are to be tied, default value is 0.

## Examples

```
stub_inst {i1} -tie 1
```

# stub_view

Removes views and related logic from a design.

## Syntax

**stub_view** {*viewName*} **-tie** {*0* | *1*}

*viewName*
Name of the view to be removed from the design. To remove more than one view, separate the names using space.

**-tie** {*0* | *1*}
The constant value with which the module pins are to be tied, default value is 0.

## Examples

```
stub_view {fsm mult} -tie 1
```

# swap_instance

Replaces the view of the specified instances in *instanceList* with the view specified by *viewName*. The interface (pin count, pin names, pin directions) of the new view must be identical to the original view. The swapped instances retain their original properties.

## Syntax

**swap_instance** *instanceList viewName*

*instanceList*
A list of the instances to be replaced.

*viewName*
The new view for the swapped instances.

# tie_net

Ties the nets from the list of nets to a constant value. The arguments can be scalar or vector names, but the widths of all arguments must match. The command assigns the same value to all connectors in the list. If the nets are vectors, the assignment is done in bit-wise manner.

## Syntax

**tie_net** *netsList* **"***value***"**

*netsList*
List of nets; the *netList* argument can have one of the forms:

*netName* or **n:***netName* – denotes a net

*instName*.*netName* or **n:***instName*.*netName* – denotes a net within the netlist of an instance

**"***value***"**
The constant value; the *value* argument can be any of the following:

**0** – changes the net driver to GND

**1** – changes the net driver to VCC

**-** (dash) – keeps the net driver unchanged

**x** or **X** – disconnects the net driver

Note that it is not necessary to quote *value* and that no spaces are allowed within the value argument string.

## Examples

The command

```
tie_net {n:KONNECT6[0]} "x"
```

disconnects the net driver from bit 0 of KONNECT6.

The commands

```
tie_net {n:KONNECT6[0:3]} "x-01"
tie_net {n:KONNECT7[5:0]} "1-x100"
```

are equivalent to the following individual tie_net commands:

```
tie_net {n:KONNECT6[0]} "x" (disconnect bit 0)
tie_net {n:KONNECT6[1]} "-" (leave bit 1 unchanged)
tie_net {n:KONNECT6[2]} "0" (set bit 2 to GND)
tie_net {n:KONNECT6[3]} "1" (set bit 3 to VCC)

tie_net {n:KONNECT7[5]} "1" (set bit 5 to VCC)
tie_net {n:KONNECT7[4]} "-" (leave bit 4 unchanged)
tie_net {n:KONNECT7[3]} "x" (disconnect bit 3)
tie_net {n:KONNECT7[2]} "1" (set bit 2 to VCC)
tie_net {n:KONNECT7[1]} "0" (set bit 1 to GND)
tie_net {n:KONNECT7[0]} "0" (set bit 0 to GND)
```

# tie_pin

Ties connectors from the list of connectors to a constant value. The arguments can be scalar or vector names, but the widths of all arguments must match. The command assigns the same value to all connectors in the list. If the connectors are vectors, the assignment is done in bit-wise manner.

## Syntax

**tie_pin** *connectorList* **"***value***"**

*connectorList*

List of connectors; the *connectorList* argument can have one of the following forms:

*instName*.*pinName* or **t:***instName*.*pinName* - denotes an instance pin

**p:***portName* - denotes a top-level output port

**p:***instName*.*portName* - denotes an output port of the internal netlist of an instance

List of connectors; the *connectorList* argument can have one of the following forms:

*instName*.*pinName* or **t:***instName*.*pinName* – denotes an instance pin

**p:***portName* – denotes a top-level output port

**p:***instName*.*portName* – denotes an output port of the internal netlist of an instance

**"***value***"**

The constant value; the *value* argument can be any of the following:

**0** – changes the net driver to GND

**1** – changes the net driver to VCC

**-** (dash) – keeps the net driver unchanged

**x** or **X** – disconnects the net driver

It is not necessary to quote *value* and that no spaces are allowed within the value argument string.

## Examples

The command

```
tie_pin {t:U6.ALARM_HRS[0]} "1"
```

ties pin ALARM_HRS[0] of instance U6 to constant value 1 (VCC).

The commands

```
tie_pin {t:U6.ALARM_HRS[0:3]} "x-01"
tie_pin {t:U6.TIME_MINS[5:0]} "1-x100"
```

are equivalent to the following individual tie_pin commands:

```
tie_pin {t:U6.ALARM_HRS[0]} "x" (disconnect pin 0)
tie_pin {t:U6.ALARM_HRS[1]} "-" (keep current connection of pin 1)
tie_pin {t:U6.ALARM_HRS[2]} "0" (tie pin 2 to constant value 0)
tie_pin {t:U6.ALARM_HRS[3]} "1" (tie pin 3 to constant value 1)

tie_pin {t:U6.TIME_MINS[5]} "1" (tie pin 5 to constant value 1)
tie_pin {t:U6.TIME_MINS[4]} "-" (keep current connection of pin 4)
tie_pin {t:U6.TIME_MINS[3]} "x" (disconnect pin 3)
tie_pin {t:U6.TIME_MINS[2]} "1" (tie pin 2 to constant value 1)
tie_pin {t:U6.TIME_MINS[1]} "0" (tie pin 1 to constant value 0)
tie_pin {t:U6.TIME_MINS[0]} "0" (tie pin 0 to constant value 0)
```

SYNOPSYS®
Silicon to Software™

CHAPTER 8

# Tcl Find and Expand

The ProtoCompiler software includes powerful search functionality in the Tcl find and expand commands. Objects located by these commands can then be grouped into collections and manipulated. A set of Synopsys® SDC commands query commands from the Design Compiler® tool are also supported for creating collections of specific object types as are Synopsys standard collection commands. The following sections describe the commands and collections in detail:

# Tcl find Command

The Tcl find command identifies and locates design objects based on a specified common characteristic. To locate objects that share connectivity, use the expand command (see *expand*, on page 391).

You can use the find command from within the constraints editor window or enter it as a Tcl command at the shell prompt. The command operates on the database and allows you to define objects identified by find as a group or *collection* and then simultaneously perform operations on all of the objects within that collection. Embedding find as part of collection creation allows the process to be completed in a single step. Combining find with collection commands provides the capability to operate on and manipulate multiple design objects simultaneously.

The table summarizes where to locate detailed information:

| For ... | See ... |
|---|---|
| Command syntax | *find (Standard)* , on page 372<br>*find* , on page 377 |
| Syntax details: object types, expressions, case sensitivity, and special characters | *Tcl Find Command Object Types (Standard)* , on page 374<br>*Tcl Find Command Object Types* , on page 379<br>*Regular Expressions, Wildcards, and Special Characters (Standard)* , on page 375<br>*Wildcards and Special Characters* , on page 380<br>*Tcl Find Command Case Sensitivity (Standard)* , on page 377<br>*Tcl Find Command Case Sensitivity* , on page 380 |
| Filtering find searches by property | *find -filter* , on page 385<br>*Find Filter Properties* , on page 385 |

## find (Standard)

Identifies design objects based on specified criteria.

## Syntax

**find**
[*-objectType*] *pattern*
[**-seq**][**-hier** [**-hsc** *character*]]
[**-flat**]
[**-in** $*collectionName*]
[**-print**]
[**-filter** *expression*]

***-objectType pattern***
Specifies the type of object to be found. Object types are view, inst, port, pin, or net. The object type must be preceded by the appropriate prefix as described in *Tcl Find Command Object Types (Standard)*, on page 374. The *pattern* argument is required and specifies the search pattern to be matched. The pattern can include the * and ? wildcard characters (see *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375).

**-seq**
Finds sequential (clocked) instances (the -inst object type is not required). This argument is equivalent to -filter @is_sequential.

**-rtl|tech**
Uses the most recently activated RTL or Technology view or opens a new one (default). **-hier** [**-hsc** *character*]
Extends the search downward through each level of the local hierarchy, instead of limiting the search to the current view. The default hierarchy separator for the search is the period (.). To specify another hierarchy character, use the -hsc option. Use this option when the pattern is ambiguous. For example, with the default separator, block1.u1 could match either instance u1 in block1 or a record named block1.u1. Using a "|" separator character eliminates the ambiguity. If you specify find -hier -hsc "|" [block1|u1], the command finds hierarchical instance u1 in the block, while find -hier --hsc "|" [block1.u1] finds the record.

**-flat**
Extends the search to all levels, but with -flat, the * wildcard character matches hierarchy separators as well as characters. This means that the following example finds instance a1_fft at the current level as well as the hierarchical instance a1.fft:

```
find -seq -flat a1*fft
```

**-in $*collectionName***

Restricts the search to the specified collection.

**-print**

Prints the first 20 search results. For a full list of objects found, use c_print or c_list. If you use find from the shell, the results are printed to the Tcl window; if you find in the constraint file, the results are printed to the log file at the beginning of the Mapper section. Reported object names have prefixes that identify the object type and double quotes around each name to allow for spaces in the names as shown below:

```
"i:reg1"
"i:\weird_name[foo$]"
"i:reg2"
<<found 233 objects. Displaying first 20 objects. Use c_print
or c_list for all. >>
```

**-filter** *expression*

Further refines the results of find by filtering the results using the specified object property. For syntax details, refer to *find -filter*, on page 385. If you use the -filter option, it must be the last option to the find command.

## Tcl Find Command Object Types (Standard)

You can specify the following types of objects:

| Object | Prefix | Example | Synopsys |
|---|---|---|---|
| view (Design) | v: | v:work.cpu.rtl is the master cell of the cpu entity, rtl architecture, compiled in the VHDL work library. | lib_cell |
| inst (Instance) | i: | Default object type. i:core.i_cpu.reg1 points to the reg1 instance inside i_cpu. | cell |
| port | p: | p:data_in[3] points to bit 3 of the primary data_in port. work.cpu.rt1\|p:rst is the hierarchical rst port in the cpu view which eventually points to all instances of cpu. | port |

| Object | Prefix | Example | Synopsys |
|---|---|---|---|
| pin | t: | t:core.i_cpu.rst points to the hierarchical rst pin of instance i_cpu. | pin |
| net | n: | n:core.i_cpu.rst points to the rst net driven in i_cpu. | net |
| seq (Sequential instance) | i: | i:core.i_cpu.reg[7:0] | cell |

## Regular Expressions, Wildcards, and Special Characters (Standard)

The Tcl find command significantly differs from a simple Tcl search. A simple Tcl search does not treat any character, except for the backslash (\), as a special character, so * matches everything in a string. The Tcl find command uses various regular expressions and special characters, as shown in the following table.

Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern, and the backslash to escape a single character.

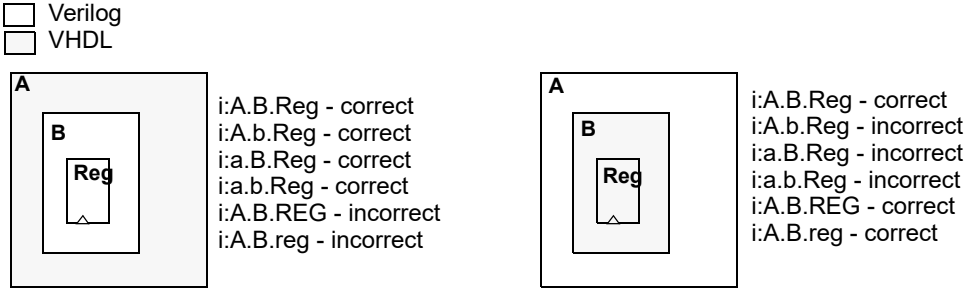
| Syntax | Matches ... |
|---|---|
| **Meta Characters: Used to match certain conditions in a string** | |
| ^ | At the beginning of the string |
| $ | At the end of the string. Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern. |
| . | Any character. If you want to use the dot (.) as a hierarchy delimiter, you must escape it with a backslash (\), because it has a special meaning in regular expressions. |
| *k* | Interprets and matches the specified non-alphanumeric character as an ordinary, non-reserved character (where *k* is the non-alphanumeric character). For example, \$ matches a dollar symbol, not the character in its reserved sense of matching the end of a string. Similarly \.d in a.b.c\.d.e indicates that c.d must be interpreted as part of the instance name, not as a hierarchy separator. |
| *c* | The specified non-alphanumeric character (where *c* is the non-alphanumeric character) when it is used in an escape sequence. |
| \| | Equivalent to an OR. |

| Syntax | Matches ... |
|---|---|
| **Character Class: A list of characters to match** | |
| [*list*] | Any single character from the *list*. For example, [abc] matches a lower-case a, b, or c. To specify a range of characters in the list, use a dash. For example, [A-Za-z] matches any alphabetical character. Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern. |
| [^*list*] | Characters not in the list. You can specify a range, as described above. For example, [^0-9] matches any non-numeric character. |
| \ | Used as a prefix to escape special characters like the following: ^ $ \ . \| ( ) [ ] { } ? + * |
| **Escape Sequences: Shortcuts for common character classes** | |
| \d | A digit between 0 and 9 |
| \D | A non-numeric character |
| \s | A white space character |
| \S | A non-white space character |
| \w | A word character; i.e., alphanumeric characters or underscores |
| \W | A non-word character |
| **Quantifiers: Number of times to match the preceding pattern** | |
| * | A sequence of 0 or more matches<br>If you do not specify -hier, the search is restricted to the current view only. To traverse downward through the hierarchy, either use the -hier argument or specify the hierarchical levels to be searched by adding the hierarchical delimiter to the pattern. For example, *.* matches objects one level below the current view. |
| + | A sequence of 1 or more matches |
| ? | A sequence of 0 or 1 matches |

| Syntax | Matches ... |
|---|---|
| {*N*} | A sequence of exactly *N* matches<br>Use curly brackets to interpret special characters as ordinary characters within a pattern. |
| {*N*,} | A sequence of *N* or more matches |
| {*N*,*P*} | A sequence of *N* through *P* matches (*P* included); *N* must be less than or equal to *P*. |

### Tcl Find Command Case Sensitivity (Standard)

Case sensitivity depends on the rules of the language used to specify the object. If the object was generated in VHDL, it is case-insensitive; if it was generated in Verilog, it is case-sensitive. In mixed-language designs, the case-sensitivity rules for the parent object prevail, even when another language is used to define the lower-level object.




## find

Finds design objects based on specified criteria.

Find is available as part of HDL Analyst.

## Syntax

**find**
[**-flat**]
[**-inst**]
[**-net**]
[**-port**]
[**-pin**]
[**-view**]
[**-nocase**]
[**-print**]
[**-depth** *value*]
[**-filter** *expression*]
[**-seq**]
[**-below**]
[*pattern*]

**-flat**
Extends the search to all levels. The * wildcard character matches hierarchy separators as well as characters. See *Regular Expressions, Wildcards, and Special Characters (Standard)*, on page 375 for additional information.

**-inst**
Finds matching instances. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

**-net**
Finds matching nets. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

**-port**
Finds matching ports. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

**-pin**
Finds matching pins. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

**-view**
Finds matching views. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

**-nocase**
The -nocase option makes the search case-insensitive.

**-print**

Prints the first 20 search results. For a full list of objects found, use c_print or c_list. If you use find from the shell, the results are printed to the Tcl window; if you find in the constraint file, the results are printed to the log file at the beginning of the Mapper section. Reported object names have prefixes that identify the object type and are contained in curly braces ( {} ).

**-depth** *value*

Sets the starting depth for the search. Value may be a single number or a range. When -depth with a range is used, for example `-depth 4-7`, -hier and -flat arguments are ignored.

**-filter** *expression*

Further refines the results of find by filtering the results using the specified object property. For syntax details, refer to *find -filter*, on page 385.

**-seq**

Finds sequential (clocked) instances (the -inst object type is not required). This argument is equivalent to -filter @is_sequential.

**-below**

Sets the start point of the search to the specified instance path. Only searches for objects below this point.

*pattern*

The value to search for.

## Tcl Find Command Object Types

You can specify the following types of objects:

| Object | Prefix | Example | Synopsys |
|---|---|---|---|
| view (Design) | v: | work.cpu.rt1\|p:rst is the hierarchical rst port in the cpu view which points to all instances of cpu. | lib_cell |
| inst (Instance) | i: | Default object type. i:core.i_cpu.reg1 points to the reg1 instance inside i_cpu. | cell |
| port | p: | p:data_in[3] points to bit 3 of the primary data_in port. work.cpu.rt1\|p:rst is the hierarchical rst port in the cpu view which eventually points to all instances of cpu. | port |

| Object | Prefix | Example | Synopsys |
|---|---|---|---|
| pin | t: | t:core.i_cpu.rst points to the hierarchical rst pin of instance i_cpu. | pin |
| net | n: | n:core.i_cpu.rst points to the rst net driven in i_cpu. | net |
| seq (Sequential instance) | i: | i:core.i_cpu.reg[7:0] | cell |

## Wildcards and Special Characters

The Tcl find command significantly differs from a simple Tcl search. A simple Tcl search does not treat any character, except for the backslash (\), as a special character, so * matches everything in a string. The Tcl find command uses various special characters, as shown in the following table.

Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern, and the backslash to escape a single character.

| Syntax | Matches ... |
|---|---|
| * | A sequence of 0 or more matches<br>If you do not specify -hier, the search is restricted to the current view only. To traverse downward through the hierarchy, either use the -hier argument or specify the hierarchical levels to be searched by adding the hierarchical delimiter to the pattern. For example, *.* matches objects one level below the current view. |
| ? | A sequence of 0 or 1 matches |

## Tcl Find Command Case Sensitivity

Case sensitivity depends on the rules of the language used to specify the object. In mixed-language designs, the case-sensitivity rules for the parent object prevail, even when another language is used to define the lower-level object.

## Tcl Find Syntax Examples

The following are examples of find syntax:

| Example | Description |
|---|---|
| find {a*} | Finds any object in the current view that starts with a |
| find {a*} -hier -nocase | Finds any object that starts with a or A |
| find -net {*synp*} -hier | Finds any net the contains synp |
| find -seq * -filter {@clock==myclk} | Finds any register in the current view that is clocked by myclk |
| find -flat -seq {U1.*} | Finds all sequential elements at any hierarchical level under U1 (* matches hierarchy separator) |
| find -hier -flat -inst {i:A.B.C.*} -filter @view==ram* | Finds all RAM instances starting from a submodule and all lower hierarchical levels from A downwards |
| find -hier -seq {*} -filter @clock_enable==ena | Finds all registers enabled by the ena signal. |
| find -hier -seq {*} -filter @slack <{-0.0} | Finds all sequential elements with negative slack |
| find -hier -seq {*} -filter {@clock ==clk1} | Finds all sequential elements within the clk1 clock domain |
| find -hier -net {*} -filter {@fanout >20} | Finds high fanout nets that drive more than 20 destinations |
| find -hier -seq * -in $all_inst_coll | Finds sequential elements inside the all_inst_coll collection |

Use the {} characters to protect patterns that contain [ ] from Tcl evaluation. For example, use the following command to find instance reg[4]:

```
find -inst {reg[4]}
```

### Find Equivalents for UCF Group Commands

The following table shows UCF commands (simple Tcl) and their equivalent find commands:

| UCF | Constraint Using find Command |
|---|---|
| INST “*ctrlfifo*" TNM = “FIFO_GRP";_ | set FIFOS [find -hier -inst {i:*ctrlfifo*}] |
| INST “*ctrlfifo" TNM = “FIFO_GRP";_ | set FIFOS [find -hier -inst {i:*ctrlfifo}] |
| INST “ctrlfifo*" TNM = “FIFO_GRP";_ | set FIFOS [find -hier -inst * -filter @hier_rtl_name == ctrlfifo*] |
| INST “ctrlfifo*hier_inst" TNM = “FIFO_GRP";_ | set FIFOS [find -hier -inst * -filter @hier_rtl_name == ctrlfifo*hier_inst] |

## Example: Custom Report Showing Paths with Negative Slack

Use the following commands:

```
view schematic implementation_a/top.srm
set find_negslack [find -hier -seq -inst {*}
   -filter @slack < {-0.0}]
c_print -prop slack -prop view $find_negslack -file negslack.txt
```

The result of running these commands is a report called negslack.txt:

```
Object Name                          slack  view
{i:CPU_A_SOC.CPU.DATAPATH.GBR[0]} -3.264  "FDE"
{i:CPU_A_SOC.CPU.DATAPATH.GBR[1]} -3.158  "FDE"
{i:CPU_A_SOC.CPU.DATAPATH.GBR[2]} -3.091  "FDE"
```

## Example: Custom Report for Negative Slack FFs in a Clock Domain

The following procedure steps through the commands used to find all negative slack flip-flops with a given clock domain:

1. Create a collection that contains all sequential elements with negative slack:

   ```
   set negFF  [find -tech -hier -seq {*} -filter @slack < {-0.0}]
   ```

2. Create a collection of all sequential elements within the clk clock domain

   ```
   set clk1FF  find -hier -seq * -filter {@clock==clk1}
   ```

3. Isolate the common elements in the two collections:

   ```
   set clk1Slack [c_intersect $negFF $clk1FF]
   ```

4. Generate a report using the c_print command:

```
c_print [find -hier -net * -filter @fanout>=2]
   {n:ack1_tmp}
   {n:ack2_tmp}
   ...
   {n:blk_xfer_cntrl_inst.lfsr_data[20:14]}
   {n:blk_xfer_cntrl_inst.lfsr_inst.blk_size[6:0]}
   {n:blk_xfer_cntrl_inst.lfsr_inst.clk_c}
   ...
```

## Custom Fanout Report Example

To generate a fanout report:

```
% c_print -prop fanout [find -hier -net * -filter @fanout>=2]
```

This is an example of the report generated by the command:

```
Object Name             Fanout
{n:ack1_tmp}            3
{n:ack2_tmp}            4
...
{n:blk_xfer_cntrl_inst.lfsr_data[14]}       3
{n:blk_xfer_cntrl_inst.lfsr_data[15]}       3
{n:blk_xfer_cntrl_inst.lfsr_data[16]}       2
...
```

You can add additional information to the report by specifying more properties. For example

```
% c_print -prop fanout [find -hier -net * -filter @fanout>=2]
   -prop pins
```

generates a report similar to the following:

```
Object Name     FanoutPins
{n:ack1_tmp}     3     "t:word_xfer_cntrl_inst.ack1_tmp
                       t:word_xfer_inst.ack1_tmp"
{n:ack2_tmp}     4     "t:blk_xfer_cntrl_inst.ack2_tmp
                       t:blk_xfer_inst.ack2_tmp"
{n:adr_o_axb_1} 2      "t:blk_xfer_inst.adr_o_axb_1
                       t:adr_o_cry_1_0.S t:adr_o_s_1.LI"
{n:adr_o_axb_2} 2      "t:blk_xfer_inst.adr_o_axb_2
                       t:adr_o_cry_2_0.S t:adr_o_s_2.LI"
{n:adr_o_axb_3} 2      "t:blk_xfer_inst.adr_o_axb_3
                       t:adr_o_cry_3_0.S t:adr_o_s_3.LI"
{n:adr_o_axb_4} 2      "t:blk_xfer_inst.adr_o_axb_4
                       t:adr_o_cry_4_0.S t:adr_o_s_4.LI"
{n:adr_o_axb_5} 2      "t:blk_xfer_inst.adr_o_axb_5
                       t:adr_o_cry_5_0.S t:adr_o_s_5.LI"
{n:adr_o_axb_6} 2      "t:blk_xfer_inst.adr_o_axb_6
                       t:adr_o_cry_6_0.S t:adr_o_s_6.LI"
...
```

To save the report as a file, use a command similar to:

```
c_print -prop fanout [find -hier -net * -filter @fanout>=2]
   -prop pins -file prop.txt
```

# find -filter

The Tcl find command includes the optional -filter option, which provides a powerful way to further refine the results of the find command and filter objects based on properties. See the following for details about the find -filter command:

- Find Filter Properties, on page 385
- Find Filter Examples, on page 389

For the Tcl find command syntax, see *find (Standard)*, on page 372.

## Syntax

**find** *pattern* [*otherOptions*] **-filter {**[**!**]**@***propertyName operator value***}**

**!**

Optional character to specify the negative. Include the ! character if you are checking for the absence of a property.

**@***propertyName*

Property name for filtering prefixed with the @ character. For example, if clock is the property name, specify {@clock==myclk}.

*operator*

Evaluates and determines the property value used for the filter expression. You can use the following operators:

Relational operators: =, <, >, ==, >=, <=

Logical operators: &&, ||, !

*value*

Property value for the property in the filter expression. The value can either be an object name such as myclk in {@clock==myclk}, or a value, such as 60 in {@fanout>=60}. If the filter expression includes spaces, the expression must be enclosed in curly braces.

## Find Filter Properties

Object properties are based on the design or constraint, and are used to qualify searches and to build collections. Some properties are only available in a specific view. The tool creates `sap` and `tap` files (design and timing properties, respectively) in the project folder.

The table below lists the common filter object properties. It does not include some vendor-specific properties. Use the table as a guide to filter the properties you want. Here is how to read the columns:

| Property Name | Property Value | Comment |
|---|---|---|
| **Common Properties** | | |
| type | view \| port \| net \| instance \| pin | |
| **View Properties** | | |
| compile_point | locked | |
| is_black_box | 1 | |
| is_verilog | 0 \| 1 | |
| is_vhdl | 0 \| 1 | |
| orig_inst_of | *viewName* | |
| syn_hier | remove \| flatten \| soft \| firm \| hard | |
| **Port Properties** | | |
| direction | input \| output \| inout | |
| fanout | *value* | Total fanout (integer) |
| side | bottom \| top\| left \| right | |
| **Instance Properties** | | |
| area | *areaValue* | |
| arrival_time | *value* | Corresponds to worst slack |
| async_reg | true \| false | |
| async_reset | **n:***netName* | |
| async_set | **n:***netName* | |
| clock | *clockName* | Could be a list if there are multiple clocks |

| Property Name | Property Value | Comment |
|---|---|---|
| clock_edge | rise \| fall \| high \| low | Could be a list if there are multiple clocks |
| clock_enable | **n:***netName* | Highest branch name in the hierarchy, and closest to the driver |
| compile_point | locked | Automatically inherited from its view |
| hier_rtl_name | *hierInstanceName* | |
| inout_pin_count | *value* | |
| input_pin_count | *value* | |
| inst_of | *viewName* | |
| is_black_box | 1 (Property added) | Automatically inherited from its view |
| is_hierarchical | 1 (Property added) | |
| is_sequential | 1 (Property added) | |
| is_combinational | 1 (Property added) | |
| is_pad | 1 (Property added) | |
| is_tristate | 1 (Property added) | |
| is_keepbuf | 1 (Property added) | |
| is_clock_gating | 1 (Property added) | |
| is_vhdl | 0 \| 1 | Automatically inherited from its view |
| is_verilog | 0 \| 1 | Automatically inherited from its view |
| kind | *primitive*<br>For example: inv\|and \|dff \| mux\|statemachine\| ...) | Mapped view contains vendor-specific primitives |
| location | (*x*, *y*) | Format can differ |
| name | *instanceName* | |

| Property Name | Property Value | Comment |
|---|---|---|
| orientation | N \| S \| E \| W | |
| output_pin_count | *value* | |
| pin_count | *value* | |
| placement_type | unplaced \| placed | |
| rtl_name | *nonhierInstanceName* | |
| slack | *value* | Worst slack of all arcs |
| slow | 1 | |
| sync_reset | **n:***netName* | |
| sync_set | **n:***netName* | |
| syn_hier | remove \| flatten \| soft \| firm \| hard | Automatically inherited from its view |
| view | *viewName* | |
| **Pin Properties** | | |
| arrival_time | *timingValue* | |
| clock | *clockName* | Could be a list if there are multiple clocks |
| clock_edge | rise \| fall \| high \| low | Could be a list if there are multiple clocks |
| direction | input \| output \| inout | |
| fanout | *value* | Total fanout (integer) |
| is_clock | 0 \| 1 | |
| is_const | 0 \| 1 | |
| is_gated_clock | 0 \| 1 | Set in addition to is_clock |
| slack | *value* | |

| Property Name | Property Value | Comment |
|---|---|---|
| **Net Properties** | | |
| clock | *clockName* | Could be a list if there are multiple clocks |
| is_clock | 0\|1 | |
| is_gated_clock | 0\|1 | Set in addition to is_clock |
| fanout | *value* | Total fanout (integer) |

## Find Filter Examples

The following examples show how find -filter is used to check for the presence or absence of a property using the ! character to indicate a negative check:

| | |
|---|---|
| `c_print [find -hier -view {*} -filter (@is_black_box)]` | Finds all objects that are black boxes. |
| `c_print [find -hier -view {*} -filter (!@is_black_box)]` | Finds all objects that are not black boxes |

### Positive Check Examples

Finds all ports, pins, and nets from the top level with a fanout greater than 8:

```
find * -filter {@fanout>8}
```

Finds all instances other than andv and orv in the design:

```
find * -hier -filter {!(@view=andv||@view=orv)}
```

Finds all instances of statemachine throughout the hierarchy:

```
find -hier -inst * -filter {@inst_of==statemachine}
```

```
find -hier -inst * -filter {@kind==statemachine}
```

Finds all instances throughout the hierarchy that include the string reg and are clocked by CLK:

```
find -hier -inst {*reg*} -filter {@clock==CLK}
```

Finds all nets throughout the hierarchy that have a fanout greater than 4:

```
find -hier -net {*} -filter {@fanout>4}
```

### Negative Check Example

Finds all instances from the top level that have the include string big, that are not black boxes, and that have more than 10 pins:

```
find -inst *big* -filter {!@is_black_box&&(@pin_count>10)}
```

### Example of Boolean Expression Specified on Multiple Properties

Finds all instances from the top level that have more than eight pins and have negative slack:

```
find * -filter {(@pin_count>8)&&(@slack<0)}
```

### Example of Pin Property Specified for Constants

Finds pins driven by constant 0 or constant 1:

```
find -pin *.* -filter @const==value -print
```

Where value of 1 lists all the pins that are tied to a constant.

### Examples of Logical OR (||) and Logical Not (!)

```
select [find -hier -inst -filter @hier_rtl_name==u1_rom ||
@hier_rtl_name==u2_rom]

select [find -hier -inst -filter !@is_combinational==0]

select [find -hier -inst -filter @pin_count==144 ||
@pin_count==15]

select [find -inst -filter @output_pin_count>10 &&
@input_pin_count>10]
```

# expand

The expand command identifies objects based on their connectivity, by expanding forward from a given starting point. For more information, see *Using the Tcl Find Command to Define Collections*, on page 71 of the *User Guide*.

## Syntax

**expand** [*-objectType*] [**-from** *object*] [**-thru** *object*] [**-to** *object*] [**-level** *integer*] [**-hier**] [**-leaf**] [**-seq**] [**-print**]

*-objectType*
Optionally specifies the type of object to be returned by the expansion. If you do not specify *objectType*, all objects are returned. The object type is one of the following:

**-inst** – returns all instances between the expansion points. This is the default.

**-pin** – returns all instance pins between the expansion points.

**-net** – returns all nets between the expansion points.

**-port** – returns all top-level ports between the expansion points.

**-from** *object*
Specifies a list or collection of ports, instances, pins, or nets for expansion forward from all listed pins. Instances and input pins are automatically expanded to all output pins of the instances. Nets are expanded to all output pins connected to the net. If you do not specify this argument, backward propagation stops at a sequential element.

**-thru** *object*
Specifies a list or collection of instances, pins, or nets for expansion forward or backward from all listed output pins and input pins respectively. Instances are automatically expanded to all input/output pins of the instances. Nets are expanded to all input/output pins connected to the net. You can have multiple -thru lists for product of sum (POS) operations.

**-to** *object*
Specifies a list or collection of ports, instances, pins, or nets for expansion backward from all the pins listed. Instances and output pins are

automatically expanded to all input pins of the instances. Nets are expanded to all input pins connected to the net. If you do not specify this argument, forward propagation stops at a sequential element.

**-level** *integer*

Limits the expansion to N logic levels of propagation. You cannot specify more than one -from, -thru, or -to point when using this option.

**-hier**

Modifies the range of any expansion to any level below the current view. The default for the current view is the top level and is defined with the define_current_design command as in the compile-point flow.

**-leaf**

Returns only non-hierarchical instances.

**-seq**

Modifies the range of any expansion to include only sequential elements. By default, the expand command returns all object types. If you want just sequential instances, make sure to define the *object_type* with the -inst argument, so that you limit the command to just instances.

**-print**

Evaluates the expand function and prints the first 20 results. If you use this command from HDL Analyst, results are printed to the Tcl window; for constraint-file commands, the results are printed to the log file at the start of the Mapper section. For a full list of objects found, you must use c_print or c_list. Reported object names have prefixes that identify the object type. There are double quotes around each name to allow for spaces in the names. For example:

```
"i:reg1"
"i:reg2"
"i:\weird_name[foo$]"
"i:reg3"
<<found 233 objects. Displaying first 20 objects. Use
   c_print or c_list for all. >>
```

## Tcl expand Syntax Examples

| Example | Description |
|---|---|
| expand -hier -from {i:reg1} -to {i:reg2} | Expands the cone of logic between two registers. Includes hierarchical instances below the current view. |
| expand -inst -from {i:reg1} | Expands the cone of logic from one register. Does not include instances below the current view. |
| expand -inst -hier -to {i:reg1} | Expands the cone of logic to one register. Includes hierarchical instances below the current view. |
| expand -pin -from {t:i_and2.z} -level 1 | Finds all pins driven by the specified pin. Does not include pins below the current view. |
| expand -hier -to {t:i_and2.a} -level 1 | Finds all instances driving an instance. Includes hierarchical instances below the current view. |
| expand -hier -from {n:cen} | Finds all elements in the transitive fanout of a clock enable net, across hierarchy. |
| expand -hier -from {n:cen} -level 1 | Finds all elements directly connected to a clock enable net, across hierarchy. |
| expand -hier -thru {n:cen} | Finds all elements in the transitive fanout and transitive fanin of a clock enable net, across hierarchy. |

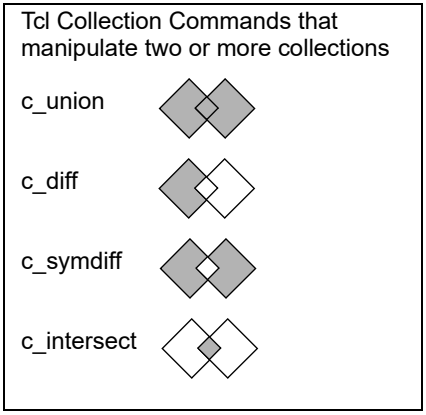
# Collection Commands

A collection is a group of objects. Grouping objects lets you operate on multiple group members at once; for example you can apply the same constraint to all the objects in a collection. You can do this from both the SCOPE editor (see Using Collections, on page 74) or in a Tcl file.

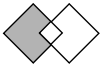
The following table lists the commands for creating, copying, evaluating, traversing, and filtering collections, and subsequent sections describe the collections, except for find and expand, in alphabetical order. For information on using collections, see *Using Collections*, on page 74 in the *User Guide*.

| Command | Description |
|---|---|
| Creation | |
| define_collection | Creates a collection from a list |
| set modules | Creates a collection |
| set modules_copy $*modules* | Copies a collection |
| Creation from Objects Identified by Embedded Commands | |
| Tcl find Command | Does a targeted search and finds objects. Embedding the find command in a collection creation command first finds the objects, and then creates a collection out of the identified group of objects. |
| expand | Identifies related objects by expanding from a selected point. Embedding the expand command in a collection creation command first finds the objects, and then creates a collection out of the identified group of objects. |
| Operators for Comparison and Analysis | |
| c_diff | Identifies differences between lists or collections |
| c_intersect | Identifies objects common to a list and a collection |
| c_symdiff | Identifies objects that belong exclusively to only one list or collection |
| c_union | Concatenates a list to a collection |

| Command | Description |
|---|---|
| Operators for Evaluation and Statistics | |
| c_info | Prints statistics for a collection |
| c_list | Converts a collection to a Tcl list for evaluation |
| c_print | Displays collections or properties for evaluation |




# c_diff



Identifies differences by comparing collections, or a list and a collection. For this command to work, the design must be open in the GUI.

## Syntax

**c_diff** {**$***collection1* **$***collection2* | **$***collection* **{***list***}**} [**-print**]

This command also includes a -print option to display the result.

### Examples

The following examples combine the set with the c_diff command to create a new collection that contains the results of the c_diff command. The first example compares two collections and puts the results in diffCollection:

```
set diffCollection [c_diff $collection1 $collection2]
```

The next example creates collection1 consisting of objects i:reg1 and i:reg2, compares this collection to a Tcl list containing object i:reg1, puts the results in the collection diffCollection and prints the result (i:reg2).

```
%set collection1 {i:reg1 i:reg2}
%set diffCollection [c_diff $collection1 {i:reg1}]
%c_print $diffCollection
{i:reg2}
```

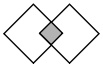
## c_info

Returns specifics of a collection, including database name, number of objects per type, and total number of objects. You can save the results to a Tcl variable (array) using the -array *name* option.

### Syntax

**c_info $***mycollection* [**-array** *name*]

## c_intersect



Defines common objects that are included in each of the collections or lists being compared.

### Syntax

**c_intersect $***collection1* **$***collection2* | *list* [**-print**]

This command also includes a -print option to display the result.

### Example

The following example uses the set command to create a new collection that contains the results of the c_intersect command. The example compares a list to a collection (myCollection) and puts the common elements in a new collection called commonCollection:

```
%set mycollection {i:reg1 i:reg2}
%set commonCollection [c_intersect $mycollection {i:reg1 i:reg3}]
%c_print $intercollection
   {i:reg1}
```

## c_list

Converts a collection to a Tcl list of objects. You can evaluate any collection with this command. If you assign the collection to a variable, you can then manipulate the list using standard Tcl list commands like lappend and lsort.

### Syntax

**c_list $***collection*|*list*

### Example

```
$set myModules [find -view *]
%c_list $myModules
{v:top} {v:block_a} {v:block_b}
```

## c_print

Displays collections or properties in column format. Object properties are printed using one or more -prop *propertyName* options.

### Syntax

**c_print** [**-append**] [**-file** *filename*] [**-foot** *outputFooter*] {[**-head** *outputHeader*] [**-prop** *propertyName*] *collection*

To print to a file, use the -file option. Use -append to append to the specified file instead of overwriting it. The following command in a constraint file prints the whole collection to a file:

```
c_print -file foo.txt $col
```

Note that the command prints the file to the active directory. You can use the `pwd` command in the Tcl window to echo the current directory and then use cd *directoryName* to change the directory as needed.

### Arguments and Options

**-append**
Appends the specified file instead of overwriting it.

**-file**
Writes the collection to *filename*.

**-foot**
Prints the output footer for the c_print command.

**-head**
Prints the output header for the c_print command.

**-prop**
Writes a column in the table for properties of type *propname*.

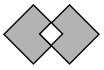
*collection*
The collection to print as a table.

### Example

```
%set modules [find -view *]
%c_print $modules
{v:top}
{v:block_a}
{v:block_b}

%c_print -prop is_vhdl -prop is_verilog $modules
Name is_vhdl is_verilog
{v:top}0 1
{v:block_a}1 0
{v:block_b}1 0
```

## c_symdiff



Compares a collection to another collection or Tcl list and finds the objects that are unique, not shared between the collections or Tcl lists being compared. It is the complement of the c_intersect command (*c_intersect*, on page 396).

### Syntax

**c_symdiff** {**$***collection1* **$***collection2* | **$***collection* **{***list***}**} [**-print**]

This command also includes a -print option to display the result.

### Examples

The following example uses the set command together with the c_symdiff command to compare two collections and create a new collection (symDiffCollection) that contains the results of the c_symdiff command.

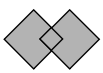
```
set symDiff_collection [c_symdiff $collection1 $collection2]
```

The next example is more detailed. It compares a list to a collection (collection1) and creates a new collection called symDiffCollection from the objects that are different. In this case, reg1 is excluded from the new collection because it is common to both the list and collection1.

```
set collection1 {i:reg1 i:reg2}
set symDiffCollection [c_symdiff $collection1 {i:reg1 i:reg3}]
c_list $symDiffCollection
   {"i:reg2" "i:reg3"}
```

You can also use the command to compare two collections:

## c_union



Adds a collection, or a list to a collection, and removes any redundant instances. For this command to work, the design must be open in the GUI.

### Syntax

**c_union** {**$***collection1* **$***collection2* | **$***collection* **{***list***}**} [**-print**]

The c_union command automatically removes redundant elements. This command also includes a -print option to display the result.

### Examples

You can concatenate two collections into a new collection using the c_union and set commands, as shown in the following example where collection1 and collection2 are concatenated into combined_collection:

```
set combined_collection [c_union $collection1 $collection2]
```

The following example creates a new collection called sumCollection, which is generated by adding a Tcl list with one object (reg3) to collection1, which consists of reg1 and reg2. The new collection created by c_union contains reg 1, reg2, and reg3.

```
%set collection1 [find -instance {reg?} -print]
   {i:reg1}
   {i:reg2}
%set sumcollection [c_union $collection1 {i:reg3}]
%c_list $sumcollection
   {i:reg1} {i:reg2} {i:reg3}
```

If instead you added reg2 and reg3 to collection1 with the c_union command, the command removes redundant instances (reg2), so that the new collection still consists of reg1, reg2, and reg3.

```
%set collection1 {i:reg1 i:reg2}
%set sumcollection [c_union $collection1 {i:reg2 i:reg3}]
%c_list $sumcollection
   {i:reg1} {i:reg2} {i:reg3}
```

## define_collection

Creates a collection from any combination of single elements, Tcl lists, and collections. You get a warning message about empty collections if you define a collection with a leading asterisk and then define an attribute for it, as shown here:

```
set noretimesh [define_collection [find -hier -seq *uc_alu]]
define_attribute {$noretimesh} {syn_allow_retiming} {0}
```

To avoid the error message, remove the leading asterisk and change `*uc_alu` to `uc_alu`.

### Example

```
set modules [define_collection {v:top} {v:cpu} $mycoll $mylist]
```

## get_prop

Returns a single property value for each member of the collection in a Tcl list.

### Examples

```
get_prop -prop clock [find -seq *]

get_prop -array arr [find A1] -all

get_prop $listExpandedInst -prop rtl_name LOROM32X1inst

get_prop $listExpandedInst -prop location SLICE_X1y36

get_prop $listExpandedInst -prop bel C6LUT

get_prop $listExpandedInst -prop slack 0.678
```

If this command is used in a Tcl script and the results need to be printed, use a puts command.

```
foreach cel [c_list $all_hier] { puts [get_prop -prop view $cel]; }
```

## set

Copies a collection to create a new collection. This command copies the collection but not the name, so the two are independent. Changes to the original collection do not affect the copied collection.

### Syntax

**set** *collectionName collectionCriteria*

**set** *copyName* **$***collectionName*

*collectionName*
The name of the new collection.

*collectionCriteria*
Criteria for defining the elements to be included in the collection. Use this argument to embed other commands, like Tcl find and expand, as

shown in the examples below, or other collection commands like define_collection, c_intersect, c_diff, c_union, and c_symdiff. Refer to the these commands for examples.

*copyName*
The name assigned to the copied collection.

**$***collectionName*
Name of an existing collection to copy.

## Examples

The following syntax examples illustrate how to use the set command. To use the set command to copy a collection:

```
set my_mod_copy $my_module
```

Use the set command with a variable name and an embedded find command to create a collection from the find command results:

```
set my_module [find -view *]
```

Use the set command with define_collection to create a collection:

```
set my_module [define_collection {v:top} {v:cpu} $col_l $mylist]
```

For more examples of the set command used with embedded Tcl collection commands, see the examples in *c_diff*, on page 395, *c_intersect*, on page 396, *c_symdiff*, on page 398, *c_union*, on page 399, and *define_collection*, on page 400.

# Object Query Commands

The query commands are Synopsys SDC commands from the Design Compiler tool for creating collections of specific object types. Functionally, they are equivalent to the Tcl find and expand commands (*Tcl find Command*, on page 372 and *expand*, on page 391).

The Synopsys SDC collection commands are only intended to be used in the FDC file to create collections of objects for constraints. This section describes the syntax for the supported object query commands. For complete documentation on these commands, refer to the Design Compiler documentation.

| | | |
|---|---|---|
| all_clocks | dot2slash | get_flat_pins |
| all_fanin | get_cells | get_nets |
| all_fanout | get_clock_source | get_pins |
| all_inputs | get_clocks | get_ports |
| all_outputs | get_flat_cells | object_list |
| all_registers | get_flat_nets | slash2dot |

## Object Query Command Syntax

All the object query commands create Tcl collections of objects for constraints, so they must be enclosed in [ ] to be applied. For example:

```
set_input_delay 0.5 [all_inputs] -clock clk
```

## Object Query Commands and Tcl find and expand Commands

The Synopsys get* and all* commands are functionally similar to the Tcl find and expand commands. The get* and all* commands are better suited for use with constraints and the FDC file as they handle properties such as @clock better than the Tcl find and expand commands. In certain cases, the FDC file does not support the find and expand commands, although you can still enter these commands in the Tcl window. See *Object Query Commands and Tcl find and expand Commands*, on page 403 for examples.

## Object Query and Tcl find/expand Examples

The following table lists parallel examples that compare how to use either the Tcl find/expand or the get/all commands to query design objects and set constraints.

| | |
|---|---|
| Return the output pins of top-level registers clocked by clkb (e.g. inst1.inst2.my_reg.Q) | |
| all_registers | FDC Constraint:<br>set_multicycle_path {4} -from [all_registers -no_hierarchy -output_pins -clock [get_clocks {clkb}]]<br>set_multicycle_path {4} -from [get_pins -of_objects [get_cells * -filter {@clock == clkb}] -filter {@name == Q}] |
| find | Tcl Window:<br>% define_collection [regsub -all {i:([^\s]+)} [join [c_list [find -inst * -filter @clock == clkfx]]] {t:\1.Q}] |
| Return all registers in the design clocked by the rising edge of clock clkfx | |
| all_registers | FDC Constraint:<br>set_multicycle_path {3} -to [all_registers -cells -rise_clock [get_clocks {clkfx}]]<br>set_multicycle_path {3} -to [get_cells -hier * -filter {@clock == clkfx && @clock_edge == rise}] |
| find | Tcl Window:<br>find -hier -inst * -filter {@clock == clkfx && @clock_edge == rise} |
| Return clock pins of all registers clocked by the falling edge of cklfx | |
| all_registers | FDC Constraint:<br>set_multicycle_path {2} -from [all_registers -clock_pins -fall_clock [get_clocks {clkfx}]]<br>set_multicycle_path {2} -from [get_pins -of_objects [get_cells -hier * -filter {@clock == clkfx && @clock_edge == fall}] -filter {@name == C}] |
| find | Tcl Window:<br>% find -hier -inst * -filter {@clock == clkfx && @clock_edge == fall} |
| Return the E pins of all instances of dffre cells (e.g. inst1.inst2.my_reg.E) | |
| get_pins | FDC Constraint:<br>set_multicycle_path -to [get_pins -filter {@name == E} -of_objects [get_cells -hier * -filter {@inst_of == dffre}] |
| find | Tcl Window and FDC Constraint:<br>% regsub -all {i:([^\s]+)} [join [c_list [find -hier -inst * -filter @inst_of == dffre]]] {t:\1.E}] |

# all_clocks

Returns a collection of clocks in the current design.

## Syntax

**all_clocks**

This command has no arguments. All clocks must be defined in the design before using this command. To create clocks, you can use the create_clock command.

## Example

The following constraint sets a multi-cycle path from all the starting points.

```
set_multicycle_path 3 -from [all_clocks]
```

# all_fanin

Use this command in the FDC constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects. Reports pins, ports, or cells for the fanin of the specified sinks in the to list.

## Syntax

**all_fanin**
[**-break_on_bboxes**]
[**-endpoints_only**]
[**-exclude_bboxes**]
[**-flat**]
[**-levels** *integer*]
[**-only_cells**]
[**-startpoints_only**]
[**-trace_arcs all**|**timing**]
**-to** *listC*

**-break_on_bboxes**
Stops timing fanin from traversing on black boxes.

**-endpoints_only**
Returns only timing end points.

**-exclude_bboxes**
Excludes black boxes from the final result.

**-flat**
The fanin function operates in flat mode. This function can be specified in hierarchical (default) or flat mode. For hierarchical mode, only objects in the same hierarchy level as the current sink are returned. The pins within a level of hierarchy below the sink are traversed, but are not reported.

**-levels** *integer*
Stops traversal when the perimeter of the search integer hops is reached. For example, a level 2 hop traverses through two levels of combinational logic and stops, instead of hopping through all levels and stopping at the first sequential or port object. Counting is performed for the layers of cells that are equidistant from the sink.

**-only_cells**
Results include a set of all cells from the timing fanin for *listC*.

**-startpoints_only**
Returns only timing start points.

**-trace_arcs all|timing**
Specifies the type of combinational arcs to trace while traversing the fanin. You can specify either:

- all – Permits tracing of all combinational arcs. This is the default.
- timing – Permits tracing of valid timing arcs only.

**-to** *listC*
*Required*. Reports a list of sink pins, ports, or nets in the design and the timing fanin of each sink in the *listC* Tcl list or collection specified. When you specify a net, effectively all drivers on that net are listed.

## Examples

The following examples show the timing fanin of a port in the design.

```
all_fanin -to [get_ports y*]
   {t:y_obuf[4].O t:y_obuf[3].O t:y_obuf[0].O t:y_obuf[1].O
   t:y_obuf[2].O t:y_obuf[5].O t:y_obuf[6].O t:y_obuf[7].O t:GND.G
   t:moduley_inst.y_c[0] t:moduley_inst.y_c[1]
   t:moduley_inst.y_c[2]}
```

```
all_fanin -to [get_ports y*] -startpoints_only -flat
   {t:moduley_inst.q[2].Q t:moduley_inst.q[1].Q
   t:moduley_inst.q[0].Q}

all_fanin -to [get_ports y*] -startpoints_only -flat -only_cells
   {i:moduley_inst.q[0] i:moduley_inst.q[1] i:moduley_inst.q[2]}
```

# all_fanout

Use this command in the FDC constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects. Returns a set of pins, ports, or cells for the fanout of the specified sources in the from list.

## Syntax

**all_fanout**
[**-break_on_bboxes**]
[**-endpoints_only**]
[**-exclude_bboxes**]
[**-flat**]
[**-levels** *integer*]
[**-only_cells**]
[**-trace_arcs all**|**timing**
**-clock_tree**|**-from** *listC*

**-break_on_bboxes**
Stops timing fanout from traversing on black boxes.

**-endpoints_only**
Returns only timing end points.

**-exclude_bboxes**
Excludes black boxes from the final result.

**-flat**
The fanout function operates in flat mode. This function can be specified in hierarchical (default) or flat mode. For hierarchical mode, only objects in the same hierarchy level as the current sink are returned. The pins within a level of hierarchy below the sink are traversed, but are not reported.

**-levels** *integer*
Stops traversal when the perimeter of the search integer hops is reached. For example, a level 2 hop traverses through two levels of

combinational logic and stops, instead of hopping through all levels and stopping at the first sequential or port object. Counting is performed for the layers of cells that are equidistant from the sink.

**-only_cells**
Results include a set of all cells from the timing fanin for *listC*.

**-trace_arcs all|timing**
Specifies the type of combinational arcs to trace while traversing the fanin. You can specify either:

- all – Permits tracing of all combinational arcs. This is the default.
- timing – Permits tracing of valid timing arcs only.

**-clock_tree|-from** *listC*
The -clock_tree argument uses all clock source pins and/or ports in the design as its list of sources. Clock sources are specified with the create_clock command. If there are no clocks or if the clocks have no sources, the report is empty. The -clock_tree option generates a report displaying the clock trees or networks in the design.

The -from argument specifies a list of source pins, ports, or nets in the design. The timing fanout for each source of the *listC* Tcl list or collection is reported. When you specify a net, effectively all load pins on the net are listed.

The -clock_tree and -from options are mutually exclusive.

## Examples

The following examples show the timing fanout of a port in the design.

```
all_fanout -from [get_ports {a*}]
   {t:a_ibuf[0].I t:a_ibuf[1].I t:a_ibuf[2].I t:hold_a.D
   t:modulex_inst.a_c[0] t:modulex_inst.a_c[1]
   t:modulex_inst.a_c[2]}

all_fanout -from [get_ports {a*}] -level 1
   {t:a_ibuf[0].I t:a_ibuf[1].I t:a_ibuf[2].I}

all_fanout -from [get_ports {a*}] -flat -endpoints_only
   {t:hold_a.D t:modulex_inst.qa[0].D t:modulex_inst.qa[1].D
   t:modulex_inst.qa[2].D t:modulex_inst.qa_fast[0].D}
```

# all_inputs

Returns a collection of input or inout ports in the current design.

## Syntax

**all_inputs**
[**-clock** *clockName*]
[**-exclude_clock_port**]

**-clock** *clockNamel*
Limits the search to ports that have input delay relative to *clockName*.

**-exclude_clock_port**
Excludes clock ports from the search.

This command has no arguments.**Examples**

The following constraints set a default input delay.

```
set_input_delay 3 [all_inputs]
set_input_delay 3 -clock {clk} [all_inputs]
```

# all_outputs

Returns a collection of output or inout ports in the current design.

## Syntax

**all_outputs**
[**-clock** *clockName*]

**-clock** *clockNamel*
Limits the search to ports that have output delay relative to *clockName*.

This command has no arguments. **Examples**

The following constraints set a default output delay.

```
set_output_delay 2 [all_outputs]
set_output_delay 2 -clock {clk} [all_outputs]
```

# all_registers

Use this command in the fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

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

## Syntax

This is the supported syntax for the all_registers command:

**all_registers**
[**-clock** *clockName*]
[**-rise_clock** *clockName*]
[**-fall_clock** *clockName*]
[**-cells**]
[**-data_pins**]
[**-clock_pins**]
[**-output_pins**]
[**-no_hierarchy**]

## Arguments

| | |
|---|---|
| **-clock** *clockName* | Searches only sequential cells that are clocked by the specified clock.<br>By default, all sequential cells in the current design are searched. |
| **-rise_clock** *clockName* | Searches only sequential cells triggered by the rising edge of the specified clock.<br>By default, all sequential cells in the current design are searched. |
| **-fall_clock** *clockName* | Searches only sequential cells triggered by the falling edge of the specified clock.<br>By default, all sequential cells in the current design are searched. |
| **-cells** | Returns a collection of sequential cells that meet the search criteria.<br>If you do not specify any of the object types, the command returns a collection of sequential cells. |

| | |
|---|---|
| **-data_pins** | Returns a collection of data pins for the sequential cells that meet the search criteria. |
| **-clock_pins** | Returns a collection of clock pins for the sequential cells that meet the search criteria. |
| **-output_pins** | Returns a collection of output pins for the sequential cells that meet the search criteria. |
| **-no_hierarchy** | Limits the search to only the current level of hierarchy. Sub-designs are not searched.<br>By default, the entire hierarchy is searched. |

### Example

The following constraint sets a max delay target for timing paths leading to all registers.

```
set_max_delay 10.0 -to [all_registers]
```

The following constraint sets a max delay target for timing paths leading to all registers clocked by PHI2.

```
set_max_delay 10.0 -to [all_registers -clock [get_clocks PHI2]]
```

## dot2slash

Translates instance strings to Xilinx place-and-route instance strings. This command is helpful for debugging your design in the HDL Analyst view. The instance string is replaced in the place-and-route tool instance string, where the:

- Dot is converted to a slash.
- Square brackets "[ ]" can be escaped if they occur in the hierarchy string, but not when they are included in the leaf portion of the string.

### Syntax

**dot2slash** *dotHierInstString*

*dotHierInstString*
Converts the synthesis instance string to a format that the place-and-route tool understands.

### Examples

In the following example, the dot is replaced by the slash in the instance string.

```
%  dot2slash {framer.go_dual\.\[9\]\.data.foo_bar[0]}
      framer/go_dual.[9].data/foo_bar[0]
```

## get_cells

Creates a collection of cells from the current design relative to the current instance.

### Syntax

**get_cells**
[**-hierarchical**]
[**-nocase**]
[**-filter** *expression*]
[*pattern*]

**-hierarchical**
Searches each level of hierarchy for cells in the design relative to the current instance. The object name at a particular level must match *pattern*. For the cell block1/adder, a hierarchical search uses "adder" to find this cell name. By default, searches are not hierarchical.

**-nocase**
Ensures that matches are case-insensitive. This applies for both the *patterns* argument and the filter operators (== and !=).

**-filter** *expressions*
Filters the collection with the specified expression. For each cell in the collection, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.

*pattern*
Creates a collection of cells whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option.

### Examples

The following example creates a collection of cells that begin with o and reference an FD2 library cell.

```
get_cells "o*" -filter "@ref_name == FD2"
```

The following example creates a collection of cells connected to a collection of pins.

```
set pinsel [get_pins o*/cp]
get_cells -of_objects $pinsel
```

The following example creates a collection of cells connected to a collection of nets.

```
set netsel [get_nets tmp]
get_cells -of_objects $netsel
```

## get_clock_source

Identifies derived clock sources when debugging your design in the HDL Analyst view. The search stops at BUFG objects. The command is not, however, as effective if clocks in the design are heavily gated as it can return all sources including registers of the gating logic.

### Syntax

**get_clock_source** *clockAlias*

*clockAlias*
Specifies the clock alias name for the synthesis clock.

### Examples

```
get_clock_source {dcm|CLK0_BUF_derived_clock_CLKIN1}
   t:dcm_inst.CLK_BUF0.O
```

## get_clocks

Creates a collection of clocks from the current design.

## Syntax

**get_clocks**
[**-nocase**]
[**-filter** *expression*]
[*pattern* |**-of_objects** *objects*]
[**-include_generated_clocks**]

**-nocase**
Ensures that matches are case-insensitive. This applies for both the *pattern* argument and the filter operators (== and !=).

**-filter** *expression*
Filters the collection with the specified expression. For each clock in the collection, the expression is evaluated based on the clock's attributes. If the expression evaluates to true, the clock is included in the result.

*pattern* |**-of_objects** *objects*
Creates a collection of clocks whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The -of_objects option creates a collection of clocks that are defined for the given net or pin objects.

**-include_generated_clocks**
Creates a collection of clocks matching the search criteria and includes any clocks derived or generated from the source clocks found. Use this syntax to propagate exception constraints through MMCM/PLL or to generated/derived clocks.

## Examples

The following example creates a collection of clocks that match the wildcard pattern.

```
get_clocks {*BUF_1*derived_clock*}
```

The following example creates a collection that includes clka and any generated or derived clocks of clka.

```
get_clocks -include_generated_clocks {clka}
```

# get_flat_cells

Creates a collection of leaf cells that match certain criteria in the current design.

## Syntax

**get_flat_cells**
[**-regexp | -exact**]
[**-nocase**]
[**-filter exper**]
[*patterns* **| -of_objects** *objects*]
[**-print**]

**-regexp**
Views the *patterns* argument as a regular expression rather than a simple wildcard pattern. This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.
When using the **-regexp** option, be careful how you quote the patterns argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" at the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

**-exact**
Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

**-nocase**
Makes matches case-insensitive, both for the patterns argument and for the ==, =~, and !~ filter operators.

**-filter** *expression*
Filters the collection with the specified expression. For each cell in the collection, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.

*patterns*
Creates a collection of cells whose full names match the specified *patterns*. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the **-nocase** option.

When using the *patterns* argument, the command searches all leaf cells to match the patterns argument on their full names regardless of their hierarchical level.

The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

**-of_objects** *objects*
Creates a collection of cells connected to the specified objects. The patterns and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

**-print**
Prints the contents of the collection.

# get_flat_nets

Creates a collection of top-level nets of hierarchical net groups in the current design that match the specified criteria.

## Syntax

**get_flat_pins**
[**-regexp | -exact**]
[**-nocase**]
[**-filter exper**]
[*patterns* **| -of_objects** *objects*]
[**-print**]

**-regexp**

Views the *patterns* argument as a regular expression rather than a simple wildcard pattern. This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.

When using the **-regexp** option, be careful how you quote the patterns argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" at the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

**-exact**

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

**-nocase**

Makes matches case-insensitive, both for the patterns argument and for the ==, =~, and !~ filter operators.

**-filter** *expression*

Filters the collection with the specified expression. For each net in the collection, the expression is evaluated based on the net's attributes. If the expression evaluates to true, the net is included in the result.

*patterns*

Creates a collection of top-level nets of hierarchical net groups whose full names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Patterns can include the asterisk (*) and question mark (?) wildcard characters. Pattern matching is case sensitive unless you use the **-nocase** option.

When using the *patterns* argument, the command searches all top-level nets of hierarchical net groups to match the patterns argument on their full name.

The patterns and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

**-of_objects** *objects*
Creates a collection of top-level nets of hierarchical net groups that are connected to the specified objects. Each object is either a named pin, port, net, cell, or a collection of these objects. The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

**-print**
Prints the contents of the collection.

# get_flat_pins

Creates a collection of leaf-cell pins that match specified criteria in the current design.

## Syntax

**get_flat_pins**
[**-regexp | -exact**]
[**-nocase**]
[**-filter exper**]
[*patterns* **| -of_objects** *objects*]
[**-print**]

**-regexp**
Views the *patterns* argument as a regular expression rather than a simple wildcard pattern. This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns.
When using the **-regexp** option, be careful how you quote the patterns argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" at the beginning or end of the expressions, as needed.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

**-exact**
Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

The **-regexp** and **-exact** options are mutually exclusive; you can use only one.

**-nocase**
Makes matches case-insensitive, both for the patterns argument and for the ==, =~, and !~ filter operators.

**-filter** *expression*
Filters the collection with the specified expression. For each pin in the collection, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

*patterns*
Creates a collection of leaf-cell pins whose full names match the specified patterns. Patterns can include the asterisk (*) and question mark (?) wildcard characters. Pattern matching is case sensitive unless you use the **-nocase** option.

When using the patterns argument, the command searches all pins of leaf cells to match the patterns argument on their full names regardless of their hierarchical level.

The patterns and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

**-of_objects** *objects*
Creates a collection of leaf-cell pins connected to the specified objects. Each object is a named leaf cell, a net, or a collection of these objects. The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.

**-print**
Prints the contents of the collection.

# get_nets

Creates a collection of nets from the current design.

## Syntax

**get_nets**
[**-hierarchical**]
[**-nocase**]
[**-filter** *expression*]
[*pattern* | **-of_objects** *objects*]

**-hierarchical**
Searches each level of hierarchy for nets in the design relative to the current instance. The object name at a particular level must match the patterns. For the net block1/muxsel, a hierarchical search uses muxsel to find this net name. By default, searches are not hierarchical.

**-nocase**
Ensures that matches are case-insensitive. This applies for both the *pattern* argument and the filter operators (== and !=).

**-filter** *expression*
Filters the collection with the specified expression. For each clock in the collection, the expression is evaluated based on the clock's attributes. If the expression evaluates to true, the clock is included in the result.

*pattern* | **-of_objects** *objects*
Creates a collection of clocks whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The -of_objects option creates a collection of clocks that are defined for the given net or pin objects.

## Examples

The following example creates a collection of nets connected to a collection of pins.

```
set pinsel [get_pins {o_reg1.Q o_reg2.Q}]
get_nets -of_objects $pinsel
```

The following example creates a collection of nets connected to the E pin of any cell in the modulex_inst hierarchy.

```
get_nets {*.*} -filter {@pins == modulex_inst.*.E}
```

# get_pins

Creates a collection of pins from the current design that match the specified criteria.

When used without -hierarchical, include a dot (.) as a pin separator between the name of the instance and the pin name. Not including the hierarchy separator results in a warning message.

## Syntax

**get_pins**
[**-hierarchical**]
[**-nocase**]
[**-filter** *expression*]
[*pattern* |**-of_objects** *objects* [**-leaf**]

**-hierarchical**

Searches each level of hierarchy for pins, relative to the current instance, and reports all instances with that pin name. By default, searches are not hierarchical.

You can use wildcards with the -hier argument. The object name at a particular level must match the pattern. For the cell block1/adder/D[0], a hierarchical search uses adder/D[0] to find pin names.

The pin separator is not required with -hier, although it is required if you use get_pins without -hier (see *Examples of get_pins*, on page 422). However, when narrowing searches by specifying instance names as well as pin names, make sure to include the hierarchy separator. Otherwise, you will not get any search results:

```
% get_pins -hier {*reset_pipe*Q}
{}
```

```
% get_pins -hier {*reset_pipe*.Q}
{t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[0].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[1].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[2].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[3].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[4].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[5].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[6].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe_0[7].Q}
```

You cannot use the -hierarchical option with the -of_objects option.

**-nocase**
Ensures that matches are case-insensitive. This applies for both the *pattern* argument and the filter operators (== and !=).

**-filter** *expressions*
Filters the collection with the specified expression. For each pin in the collection, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

*pattern* | **-of_objects** *objects* [**-leaf**]
Creates a collection of pins whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The -of_objects option creates a collection of pins connected to the specified objects. Each object can be a cell or net. By default, the command considers only pins connected to the specified nets at the same hierarchical level. To consider only pins connected to leaf cells on the specified nets, use the -leaf option (the tool can cross hierarchical boundaries to find pins on leaf cells). You cannot use the -hierarchical option with the -of_objects option.

## Examples of get_pins

This example creates a collection of all pins in the design.

```
get_pins -hier *.*
```

This example shows that without a separator, the command returns no results and generates a warning message:

```
% get_pins {Q}

Warning: No pin separator ('.') specified. Pattern must include a
pin separator.
```

The following example creates a collection of pins throughout the hierarchy that match the regular expression.

```
get_pins -hier - regexp {.*\.ena}
```

This example illustrates that you do not need the pin separator when you specify the -hier argument:

```
% get_pins -hier {Q}

{t:haps_system_capim.capi_di[0].Q t:haps_system_capim.capi_di[1].Q
t:haps_system_capim.capi_di[2].Q t:haps_system_capim.capi_di[3].Q
t:haps_system_capim.capi_di[4].Q t:haps_system_capim.capi_di[5].Q
t:haps_system_capim.capi_di[6].Q t:haps_system_capim.capi_di[7].Q
t:haps_system_capim.capi_di[8].Q t:haps_system_capim.capi_di[9].Q
t:haps_system_capim.capi_di[10].Q
t:haps_system_capim.capi_di[11].Q
...}
```

The next example creates a collection of hierarchical pin names for the library cell pin DQSFOUND, and for each instantiation of a library cell named PHASER_IN_PHY.

```
get_pins -filter {@name == DQSFOUND} -of_objects [get_cells -hier
* -filter {@inst_of == PHASER_IN_PHY}]
```

# get_ports

Creates a collection of top-level ports from that match the specified criteria.

## Syntax

**get_ports**
[**-nocase**]
[**-filter** *expression*]
[*pattern*]

**-nocase**
Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (== and !=).

**-filter** *expressions*
Filters the collection with the specified expression. For each port in the collection, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the port is included in the result.

*pattern*

Creates a collection of ports whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The pattern and -of_objects arguments are mutually exclusive. If you do not specify either argument, the command uses * (asterisk) as the default pattern.

### Examples

The following example queries all input ports beginning with mode.

```
get_ports mode* -filter {@direction == input}
```

## object_list

Translates object strings returned by query commands in the HDL Analyst tool to proper Tcl lists. Using this command allows you to process the results using Tcl commands.

### Syntax

**object_list** *objectString*

*objectString*

Converts the object string returned by an FDC query command to proper Tcl lists.

### Examples

```
% foreach x [object_list [get_cells -hier {q[*]}]]
      {puts "Match: $x"}

Match: i:modulex_inst.q[7:0]
Match: i:moduley_inst\[4\].q[7:0]
```

## slash2dot

Translates Xilinx place-and-route instance strings to ProtoCompiler instance strings. This command is helpful for debugging your design in the HDL Analyst view. The place-and-route instance string is replaced in the Proto-Compiler instance string, where the:

- Slash is converted to a dot.
- Square brackets "[ ]" can be escaped if they occur in the hierarchy string, but not when they are included in the leaf portion of the string.

## Syntax

**slash2dot** *slashHierInstString*

*slashHierInstString*
Converts the place-and-route instance string to a format that the Proto-Compiler understands.

## Examples

In the following example, the slash is replaced by the dot in the instance string.

```
%  slash2dot {framer/go_dual.[9].data/foo_bar[0]}
      framer.go_dual\.\[9\]\.data.foo_bar[0]
```

# Synopsys Standard Collection Commands

There are a number of Synopsys standard SDC collection commands that can be included in the FDC file. These commands are not compatible with the define_scope_collection command.

The collection commands let you manipulate or operate on multiple design objects simultaneously by creating, copying, evaluating, iterating, and filtering collections. This section describes the syntax for the following collection commands supported in the FPGA synthesis tools; for the complete syntax for these commands, refer to the Design Compiler documentation.

- add_to_collection
- append_to_collection
- copy_collection
- foreach_in_collection
- get_object_name
- index_collection
- remove_from_collection
- sizeof_collection

Use these commands in the FDC constraint file to facilitate the shared scripting of constraint specifications between the FPGA synthesis and Design Compiler tools.

## add_to_collection

Adds objects to a collection that results in a new collection. The base collection remains unchanged.

### Syntax

**add_to_collection** *collection1 objectSpec*

*collection1*
Specifies the base collection to which objects are to be added. This collection is copied to a resulting collection, where objects matching *objectSpec* are added to this results collection.

*objectSpec*

Specifies a list of named objects or collections to add. Depending on the base collection type (heterogeneous or homogeneous), the searches and resulting collection may differ. For more information, see *Heterogeneous Base Collection*, on page 427 and *Homogeneous Base Collection*, on page 427.

## Description

The add_to_collection command allows you to add elements to a collection. The result is a new collection representing the objects added from the *objectSpec* list to the base collection. Any duplicate objects in the resulting collection are automatically removed from the collection. If *objectSpec* is empty, then the new collection is a copy of the base collection. Depending on the base collection type (heterogeneous or homogeneous), the searches and resulting collection may differ.

### Heterogeneous Base Collection

If the base collection is heterogeneous, then only collections are added to the resulting collection. All implicit elements of the *objectSpec* list are ignored.

### Homogeneous Base Collection

If the base collection is homogeneous and any elements of *objectSpec* are not collections, then the command searches the design using the object class of the base collection.

When *collection1* is an empty collection, special rules apply to *objectSpec*. If *objectSpec* is not empty, at least one homogeneous collection must be in the *objectSpec* list (can be any position in the list). The first homogeneous collection in the *objectSpec* list becomes the base collection and sets the object class for the function.

## Example

```
set result [get_cells{u*}]
get_object_name $result

==> {u:u1} {i:u2} {i:u3}

set result_1 [add_to_collection $result {get_cells {i:clkb_IBUFG}]
get_object_name $result_1
```

```
==> {i:u1} {i:u2} {i:u3} {i:clkb_IBUFG}
```

### See Also

- append_to_collection

## append_to_collection

Adds objects to the collection specified by a variable, modifying its value. Objects must be unique, since duplicate objects are not supported.

### Syntax

**append_to_collection** *variableName objectSpec*

*variableName*
Specifies a variable name. The objects matching *objectSpec* are added to the collection referenced by this variable.

*objectSpec*
Specifies a list of named objects or collections to add to the resulting collection.

### Description

The append_to_collection command allows you to add elements to a collection. This command treats the *variableName* option as a collection, and appends all the elements of *objectSpec* to that collection. If the variable does not exist, it creates a collection with elements from the *objectSpec* as its value. So, a collection is created that was referenced initially by *variableName* or automatically if the *variableName* was not provided. However, if the variable exists but does not contain a collection, then an error is generated.

The append_to_collection command can be more efficient than the add_to_collection command (*add_to_collection*, on page 426) when you are building a collection in a loop.

### Example

```
set result [get_cells{u*}]
get_object_name $result
```

```
==> {u:u1} {i:u2} {i:u3}

append_to_collection result {get_cells {i:clkb_IBUFG}]
get_object_name $result

==> {i:u1} {i:u2} {i:u3} {i:clkb_IBUFG}
```

### See Also

- add_to_collection

## copy_collection

Duplicates the contents of a collection that results a new collection. The base collection remains unchanged.

### Syntax

**copy_collection** *collection*1

*collection1*
Specifies the collection to be copied.

### Description

The copy_collection command is an efficient mechanism to create a duplicate of an existing collection. It is sometimes more efficient and usually sufficient to simply have more than one variable referencing the same collection. However, whenever you want to copy the collection instead of referencing it, use the copy_collection command.

Be aware that if an empty string is used for the *collection1* argument, the command returns an empty string. This means that a copy of the empty collection is an empty collection.

### Example

```
set insts [define_collection {u1 u2 u3 u4}]
set result_copy [copy_collection $insts]
get_object_name $result_copy

==> {u1} {u2} {u3} {u4}
```

# foreach_in_collection

Iterates on the elements of a collection.

## Syntax

**foreach_in_collection** *iterationVariable collections body*

*iterationVariable*
Specifies the name of the iteration variable. It is set to a collection of one object. Any argument that accepts collections as an argument can also accept the *iterationVariable*, as they are the same data type.

*collections*
Specifies a list of collections on which to iterate.

*body*
Specifies a script to execute for the iteration. If the body of the iteration is modifying the netlist, all or part of the collection involved in the iteration can be deleted. The foreach_in_collection command is safe for such operations. A message is generated that indicates the iteration ended prematurely.

## Description

The foreach_in_collection command iterates on each element of a collection. This command requires the following arguments: an iteration variable (do not specify a list), the collection on which to iterate, and the script to apply for each iteration.

You can nest this command within other control structures, including another foreach_in_collection command.

You can include the command in an FDC file, but if you are using the Tcl window and HDL Analyst, you must use the standard Tcl foreach command instead of foreach_in_collection.

## Example

The following examples show valid methods to reference a collection for this command:

```
set seqs[all_registers]
set port[all_inputs]
```

```
foreach_in_collection x [all_registers] {body}
foreach_in_collection x $ports {body}
foreach_in_collection x [list $seqs $ports] {body}
foreach_in_collection x {$seqs} {body}
foreach_in_collection x {$seqs $ports} {body}
```

# get_object_name

Returns a list of names for objects in a collection.

## Syntax

**get_object_name $***collectionName*

**$***collectionName*
Specifies the name of the collection that contains the requested objects.

## Example

```
set c1[define_collection {u1 u2}]
get_object_name $c1
```

```
==> {u1} {u2}
```

# index_collection

Creates a new collection that contains only the single object for the index specified in the base collection. You must provide an index to the collection.

## Syntax

**index_collection** *collection1 index*

*collection1*
Specifies the collection to be searched.

*index*
Specifies an index to the collection. Allowed values are integers from 0 to *sizeof_collection* -1.

### Description

You can use the index_collection command to extract a single object from a collection. The result is a new collection that contains only this object. The range of indices can be from 0 to one less than the size of the collection. If the specified index is outside that range, an error message is generated.

Commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same, predictable order each time. Applications that support the sorting of collections allow you to impose a specific order on a collection.

If you use an empty string for the *collection1* argument, then any index to the empty collection is not valid. This results in an empty collection and generates an error message.

Be aware that all collections cannot be indexed.

### Example

```
set c1[get_cells {u1 u2}]]
get_object_name [index_collection $c1 0]

==> {u1}
```

### See Also

- sizeof_collection

## remove_from_collection

Removes objects from a collection that results in a new collection. The base collection remains unchanged.

### Syntax

This is the supported syntax for the remove_from_collection command:

**remove_from_collection** [**-intersect**] *collection1 objectSpec*

**-intersect**

Removes objects from the base collection that are not found in *objectSpec*. By default, when this option is not specified, objects are removed from the base collection that are found in the *objectSpec*.

*collection1*

Specifies the base collection that is copied to a resulting collection, where objects matching *objectSpec* are removed from this results collection.

*objectSpec*

Specifies a list of named objects or collections to remove. The object class for each element in this list must be the same in the base collection. If the name matches an existing collection, that collection is used. Otherwise, objects are searched in the design using the object class for the base collection.
Description

The remove_from_collection command removes elements from a collection and creates a new collection. When the -intersect option is not specified and there are no matches for *objectSpec*, the resulting collection is just a copy of the base collection. If everything in *collection1* matches *objectSpec*, the result is an empty collection. When using the -intersect option, nothing is removed from the resulting collection.

### Heterogeneous Base Collection

If the base collection is heterogeneous, then any elements of *objectSpec* that are not collections are ignored.

### Homogeneous Base Collection

If the base collection is homogeneous and any elements of *objectSpec* are not collections, then the command searches the design using the object class of the base collection.

## Examples

```
set c1[define_collection {u1 u2 u3}]]
set c2[define_collection {u2 u3 u4}]]
get_object_name [remove_from_collection $c1 $c2]

==> {u1}

get_object_name [remove_from_collection $c2 $c1]
```

```
==> {u4}

get_object_name [remove_from_collection -intersect $c1 $c2]

==> {u2} {u3}
```

### See Also

- add_to_collection

## sizeof_collection

Returns the number of objects in a collection.

### Syntax

**sizeof_collection** *collection1*

*collection1*
Specifies the name of the collection for which the number of objects is requested. If no collection argument is specified, then the command returns 0.

### Examples

```
set c1[define_collection {u1 u2 u3}]
sizeof_collection $c1

==> 3
```

SYNOPSYS®
Silicon to Software™

CHAPTER 9

# Commands for Unified Compile

The following commands and syntax are used in the Unified Compile (UC) flow, but are commands outside the scope of the tool. Commands for the UC flow from within the tool, such as launch uc, are described in *Shell Command Reference*, on page 17.

# SystemVerilog $dumpvars Syntax

This is a summary description. For comprehensive information, refer to the SystemVerilog documentation.

$dumpvars is a SystemVerilog construct that you can use to specify probe signals. You can use the $dumpvars task with or without arguments. For example:

```
$dumpvars ;
| $dumpvars ( levels [ , list_of_modules_or_variables ] ) ;
   list_of_modules_or_variables ::=
      module_or_variable { , module_or_variable }
      module_or_variable ::=
         module_identifier
         | variable_identifier
```

This is an RTL example:

```
initial
begin: IICE_0
   $dumpvars (1, top.inst1.sig1);
end

initial
   begin: IICE_1
      $dumpvars (1, top.inst2.sig2);
end
```

The $dumpvars arguments specify the granularity of what is dumped. The first argument indicates how many *levels* of hierarchy below the specified module instance must be dumped to the VCD file. Subsequent arguments specify the scope of the model to dump. These arguments can be used to specify entire modules or individual variables within a module. When invoked with no arguments, $dumpvars dumps all the variables in the model to the VCD file.

Note that $dumpvars (0,<top_module>) is not supported; you cannot dump all levels of hierarchy, especially in large designs.

The VHDL equivalent of $dumpvars is $dumpport.

# UTF Commands

The UTF file contains commands for running VCS synthesis. See the VCS documentation for comprehensive information about the syntax. The following topics provide brief descriptions of some commands used in the UTF file.



## assertion_synthesis Syntax

The UTF assertion_synthesis command controls how SVAs are implemented. Refer to the VCS documentation for the complete syntax.

**assertion_synthesis** [**-enable** <*value*>] [**-ignore** (<*sva_type*>)] [**-path** <*path*>] [**-tree** <*hierarchy*>] [**-module** <*module_name*>] [–**assert** <*#SVA_instance*>]

| | |
|---|---|
| assertion_synthesis -enable ALL | Synthesizes all assertions. |
| assertion_synthesis -ignore IMMEDIATE \| CONCURRENT | Specifies the type of assertions to ignore. |
| assertion_synthesis –path <*hierarchy_name*> | Does not synthesize SVAs in the given hierarchy. |
| assertion_synthesis +path <*hierarchy_name*> | Synthesizes all SVAs in the specified hierarchy. |
| assertion_synthesis –tree <*hierarchy_name*> | Does not synthesize SVAs below the given hierarchy. |
| assertion_synthesis +tree <*hierarchy_name*> | Synthesizes all SVAs below the given hierarchy. |
| assertion_synthesis –module <*module_name*> | Does not synthesize any SVAs in the given module. |
| assertion_synthesis +module <*module_name*> | Synthesizes all SVAs in the given module. |
| assertion_synthesis -ignore ALL | Does not synthesize any SVAs. |
| assertion_synthesis –assert *#SVA_instance* | Does not synthesize the specified SVA instance. |
| assertion_synthesis +assert *#SVA_instance* | Synthesizes the specified SVA instance. |

## wire_resolution Command for XMR Conflicts

The UTF wire_resolution command is used to resolve conflicts when there are multiple drivers. This is the RTL code:

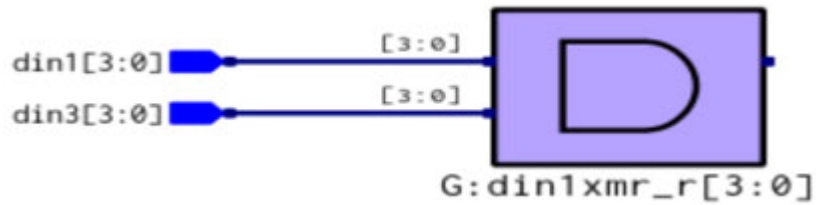
```
assign top.inst1.din1=din3;
sub1 #(.width(width)) inst1
(.clk(clk),.rst_n(rst_n),.din1(din1),.dout1(dout1));
```

This is the UTF syntax:

```
wire_resolution -default_xmr_conflict {WAND | WOR | XMR}
```

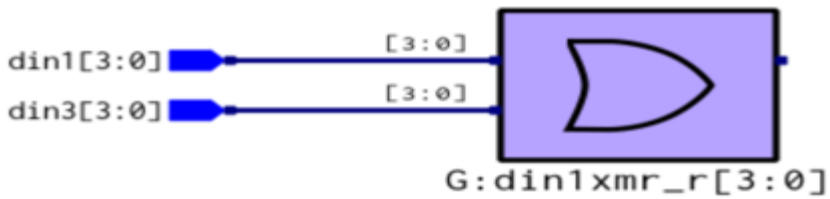
The following examples show the effects of different wire_resolution command settings:

- Implementation when wire_resolution is set to the default, WAND:




wire_resolution -default_xmr_conflict {WAND}

- Implementation when wire_resolution is set to WOR:




wire_resolution -default_xmr_conflict {WOR}

- When wire_resolution is set to XMR, the XMR has the highest priority.

# vcs Command

This command is included in the VCS script and runs the VCS front-end compiler (UC) on the HDL design.

Refer to the VCS documentation for the complete syntax for this command; only some of the arguments are described here.

## vcs Command Syntax for Elaboration

**vcs** <*files_to_compile*> **-l** *logfile* **-P pli.tab -xlrm -ignore driver_checks -liblist:**

These command arguments build the design hierarchy from the library files generated during the analysis phase (vlogan and vhdlan commands in the VCS script). The table below lists commonly used arguments.

| vcs Argument | Description |
|---|---|
| -l <*logfile*> | Generates the log file |
| -l <*logfile*> | Generates the log file |
| -P pli.tab | Compiles the user-defined PLI table |
| <*.c*\|*.o files*> | Adds C or object files to compile |
| -xlrm | Allows a relaxed/non-LRM compliant code |
| -ignore driver_checks | Suppresses multiple driver checks |
| -liblist: | Specifies the library search order for unresolved |
| module or entity | Instantiated in Verilog |

## vcs Syntax for Specifying a UPF File

**vcs -upf** *designtop.upf* **-power=unified_model -power_top** *designtop*
**[-power=scm_mem_ret]**
**[-power=seqcorrupt] [-power=hw_corrupt_memory]**
**[-power=enable_boundary_gates]**

| | |
|---|---|
| -upf *designtop*.upf | Required. Specify this switch to source the DUT top-level UPF file. |
| -power=unified_model | Required. Specify this switch to enable retention of flops. |
| -power_top=*designtop* | Required when *designtop*.upf file does not contain the set_design_top *designtop* command. |
| [-power=scm_mem_ret] | Optional. Specify to enable retention of memory. |
| [-power=seqcorrupt] | Optional. Specify to enable flop corruption. |
| [-power=hw_corrupt_memory] | Optional. Specify to enable memory corruption. You can then specify a corruption value by adding an attribute in the UPF file. |
| [-power=enable_boundary_gates] | Optional. Specify this switch to enable boundary mux-based corruption (fixed value 0). |

See *UPF File for Unified Compile*, on page 188 in the *Reference Manual* for additional information about the contents of the file.

## vcs Command Syntax for Information

Use these vcs commands to get information:

| Command | Description |
|---|---|
| vcs -full64 -id | Checks the VCS setup, version, and platform |
| vcs -full64 -doc | Accesses VCS setup and compilation commands% |

# zFmCheck Command

Checks elaborated RTL against the output of Simon synthesis, using the VCS simulation or Formality tools.

## Syntax

**zFmCheck**
**[-scm]**
**[-scm_sim]**
**[-d** *designDir***]**
**[-g | -o** *outputDir***]**
**[-l** *hrs*:*mins*:*secs***]**
**[-j** *parallel_jobs***]**
**[-t** *top_module***]**
**[-m** *subModule***]**
**[-mf** *Tcl_file***]**
**[-mh** *subModule_with_hierarchy***]**
**[-Verdi_debug]**
**[-help]**

**-d** *designDir*
Specifies the design directory to validate.

**-g | -o** *outputDir*
Specifies the output directory for results from the run.

**-help**
Prints syntax information for the command.

**-j** *parallel_jobs*
Specifies the number of parallel jobs to use for the validation run.

**-l** *hrs*:*min*:*seconds*
Specifies the time for the validation run. For example: -l 01:00:00.

**-m** *subModule*
Specifies the submodule to be validated.

**-mf** *Tcl_file*
Specifies a file with a list of modules to be validated.

**-mh** *subModule_with_hierarchy*
Specifies the hierarchical submodule to be validated.

**-scm**
Runs Formality equivalence checking to compare the RTL to the Simon results. Either this argument or -scm_sim is required.

**-scm_sim**
Runs VCS simulation to compare the RTL to the Simon results. If both -scm and -scm_sim are specified, simulation is run when the formal verification run fails. Either this argument or -scm is required.

**-t** *top_module*
Specifies the top module for the design.

**-Verdi_debug**
Runs Verdi debug on issues that failed in simulation.

## Description

Run this command from a terminal prompt. The command runs the comparison using simulation or equivalence checking, and generates a log file and a readme.txt file in the specified output directory.

## Examples

- Validate a module using only Formality equivalence checking:

```
zFmCheck -scm  -d ucdb/design/defaultGroup/ -g submodule1
–l 01:00:00 –j 16 –m myModule
```

- Validate the hierarchy of a module and its instances:

```
zFmCheck -scm  -scm_sim -d ucdb/design/defaultGroup/ -o
module_hier –l 01:00:00 –j 16 –mh mySubModule
```

- Run simulation only and debug all failures and errors with the Verdi software:

```
zFmCheck  -scm_sim -d ucdb/design/defaultGroup/ -g failure_list
-l 01:00:00 -j 16 –mf sub_module.tcl –Verdi_debug
```

# Index



## Symbols



## A



## B



## C

## D



## E



## F

## G



## H



## I



## J



## L



## M



## N



## O

## P



## Q



## R



## S

## T



## U

## V



## W