Formality®
Tool Commands

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice for the Line-Editing Library

Contents

add_to_collection

Adds objects to a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection add_to_collection
   [-unique]
   collection1
   object_spec
```

Data Types

```
collection1          collection
object_spec          list
```

ARGUMENTS

-unique

Removes duplicate objects from the resulting collection. By default, duplicate objects are not removed.

collection1

Specifies the base collection to which objects are to be added. This collection is copied to the result collection, and objects matching *object_spec* are added to the result collection. The *collection1* option can be the empty collection (empty string), subject to some constraints, as explained in the DESCRIPTION section.

object_spec

Specifies a list of named objects or collections to add.

If the base collection is heterogeneous, only collections can be added to it.

If the base collection is homogeneous, the object class of each element in this list must be the same as in the base collection. If it is not the same class, it is ignored. From heterogeneous collections in the *object_spec*, only objects of the same class of the base collection are added. If the name matches an existing collection, the collection is used. Otherwise, the objects are searched for in the database using the object class of the base collection.

The *object_spec* has some special rules when the base collection is empty, as explained in the

DESCRIPTION section.

DESCRIPTION

The **add_to_collection** command allows you to add elements to a collection. The result is a new collection representing the objects in the *object_spec* added to the objects in the base collection.

Elements that exist in both the base collection and the *object_spec*, are duplicated in the resulting collection. Duplicates are not removed unless you use the *-unique* option. If the *object_spec* is empty, the result is a copy of the base collection.

If the base collection is homogeneous, the command searches in the database for any elements of the *object_spec* that are not collections, using the object class of the base collection. If the base collection is heterogeneous, all implicit elements of the *object_spec* are ignored.

When the *collection1* argument is the empty collection, some special rules apply to the *object_spec*. If the *object_spec* is non-empty, there must be at least one collection somewhere in the *object_spec* list (its position in the list does not matter). The first homogeneous collection in the *object_spec* list becomes the base collection and sets the object class for the function. The examples show the different errors and warnings that can be generated.

The **append_to_collection** command has similar semantics as the **add_to_collection** command; however, the **append_to_collection** command can be much more efficient in some cases. For more information about the command, see the man page.

For background on collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example uses the **get_ports** command to get all of the ports beginning with 'mode' and then adds the "CLOCK" port.

```
prompt> set xports [get_ports mode*]
{r:/WORK/top/mode[0] r:/WORK/top/mode[1] r:/WORK/top/mode[2]}
prompt> add_to_collection $xports [get_ports CLOCK]
{r:/WORK/top/mode[0] r:/WORK/top/mode[1] r:/WORK/top/mode[2] r:/WORK/top/CLOCK}
```

The following example adds the cell u1 to a collection containing the SCANOUT port.

```
prompt> set so [get_ports SCANOUT]
{r:/WORK/top/SCANOUT}
prompt> set u1 [get_cells u1]
{r:/WORK/top/u1}
prompt> set het [add_to_collection $so $u1]
{r:/WORK/top/SCANOUT r:/WORK/top/u1}
prompt> query_objects -verbose $het
{{port r:/WORK/top/SCANOUT} {cell r:/WORK/top/u1}}
```

The following examples show how the **add_to_collection** command behaves when the base collection is empty. Adding two empty collections yields the empty collection. Adding an implicit list of only strings to the empty collection generates an error message, because no collections are present in the *object_spec* list. Finally, as long as one collection is present in the *object_spec* list, the command succeeds, even though a warning message is generated. The example uses the variable settings from the previous example.

```
prompt> sizeof_collection [add_to_collection "" ""]
0

prompt> set A [add_to_collection "" [list a c]]
Error: At least one collection required for argument 'object_spec'
        to add_to_collection when the 'collection' argument is empty (SEL-014)

prompt> add_to_collection "" [list a $het $so]
Warning: Ignored all implicit elements in argument 'object_spec'
         to add_to_collection because the class of the base collection
         could not be determined (SEL-015)
{r:/WORK/top/SCANOUT r:/WORK/top/u1 r:/WORK/top/SCANOUT}
```

SEE ALSO

```
append_to_collection(2)
collections(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
```

alias

Creates a pseudocommand that expands to a longer command with more words, or lists the current alias definitions.

SYNTAX

```
alias
   [name]
   [def]
```

Data Types

```
name string
def string
```

ARGUMENTS

name

Specifies the name of the alias. The name must begin with a letter, and can contain letters, underscores, and numbers.

def

Specifies the expansion of the alias. That is, it specifies the replacement text, for the *name* argument.

DESCRIPTION

This command defines or displays command aliases.

When you don't specify any arguments, it displays the currently defined aliases and their expansions. With a single argument, the **alias** command displays the expansion for the given *name*. With more than one argument, an alias is created that is named by the first argument and expanded to the remaining arguments.

You cannot create an alias by using the name of any existing command or procedure. Thus, you cannot use the **alias** command to redefine existing commands.

An alias can refer to another alias.

An alias is expanded only when it is the first word in a command.

EXAMPLES

Although you can abbreviate commands, there could be a conflict with another command. The following example shows how to use an alias to prevent the conflict.

```
prompt> alias q quit
prompt>
```

The following example shows how to use the **alias** command to create a shortcut for commonly used command invocations.

```
prompt> alias include {source -echo -verbose}
prompt>
```

After the previous commands, the **include script.Tcl** is replaced with **source -echo -verbose script.Tcl** before the command is interpreted. The following examples show how to display the alias by using the **alias** command. Note that the displayed aliases are in alphabetical order.

```
prompt> alias
include        source -echo -verbose
q              quit
prompt>
```

SEE ALSO

```
unalias(2)
```

all_connected

Returns the objects connected to a net, port, or pin.

SYNTAX

```
collection all_connected
   [-leaf]
   [-type type]
   [objects]
```

Data Types

```
   type           string
   objects        list
```

ARGUMENTS

-leaf

Specifies that only leaf pins are returned for a hierarchical net. For non-hierarchical nets, there is no difference in output.

-type *type*

The type of the *objects*. The type can be net, pin, or port.

objects

Specified the objects whose connections are returns. The objects must be nets, port, or pins. Objects of other types are ignored.

DESCRIPTION

The **all_connected** command returns a collection of objects connected to the specified nets, ports, and pins.

If the *-leaf* option is sued, a collection of leaf pins and ports of the nets is returned.

To connect nets to ports or pins, use the **connect_net** command. TO break connections, use the **disconnect_net** command.

EXAMPLES

The following example uses **all_connected** to return the objects connected to **MY_NET**:

```
prompt> all_connected MY_NET

# Connecting net 'MY_NET' to port 'OUT3'.
prompt> connect_net MY_NET OUT3

# Connecting net 'MY_NET' to pin 'U65/Z'.
prompt> connect_net MY_NET U65/Z

prompt> all_connected MY_NET
{r:/WORK/top/OUT3 r:/WORK/top/U65/Z}

prompt> all_connected OUT
{r:/WORK/top/MY_NET}

prompt> all_connected U65/Z
{r:/WORK/top/MY_NET}
```

SEE ALSO

```
all_inputs(2)
all_outputs(2)
connect_net(2)
create_net(2)
current_design(2)
disconnect_net(2)
remove_net(2)
```

all_fanin

Returns a collection of pins, ports, or cells in the fanin of the specified sinks.

SYNTAX

```
collection all_fanin
   [-type type]
   [-to objects]
   [-only_cells]
   [-flat]
```

Data Types

```
   type          string
   objects       list
```

ARGUMENTS

-type *type*

The type of the *objects*. The type can be net, pin, port, or cell.

-to *objects*

Returns a collection of sink pins and port in the design.

-only_cells

Returns a collection of all cells in the fanin of *objects*, rather than a collection of pins and ports.

-flat

Specifies to function in the flat mode of operation. The two major modes in which **all_fanin** functions are hierarchical (the default) and flat. When in hierarchical mode, only objects from the same hierarchy levels as *objects* are returned.

DESCRIPTION

The **all_fanin** command returns a collection of the fanin of specified sink objects in the design. The command stops at inputs to logic cones: For example, registers, cut-points, and black-box output pins.

EXAMPLES

The following examples show the fanin of a port in the design. THe design comprises three inverters in a chain name *iv1*, *iv2*, and *iv3*. The *iv1* and *iv2* inverters are hierarchically combined in a larger cell named *ii2*.

To make it easier to read, we have ommitted the *container:/library/design* from the output below.

```
prompt> all_fanin -to tout
{tin ii2/hin ii2/hout iv3/in iv3/out tout}

prompt> all_fanin -to tout -flat
{tin ii2/iv1/U1/a ii2/iv1/U1/z ii2/iv2/U1/z ii2/iv2/U1/a iv3/U1/a iv3/U1/z tout}
```

SEE ALSO

all_fanout(2)

all_fanout

Returns a collection of pins, ports, or cells in the fanout of the specified sources.

SYNTAX

```
collection all_fanout
   [-type type]
   [-from objects]
   [-only_cells]
   [-flat]
```

Data Types

```
type            string
objects         list
```

ARGUMENTS

-type *type*

The type of the *objects*. The type can be net, pin, port, or cell.

-from *objects*

Returns a collection of source pins and port in the design.

-only_cells

Returns a collection of all cells in the fanout of *objects*, rather than a collection of pins and ports.

-flat

Specifies to function in the flat mode of operation. The two major modes in which **all_fanout** functions are hierarchical (the default) and flat. When in hierarchical mode, only objects from the same hierarchy levels as *objects* are returned.

DESCRIPTION

The **all_fanout** command returns a collection of the fanout of specified source objects in the design. The command stops at outputs of logic cones: For example, registers, cut-points, and black-box input pins.

EXAMPLES

The following examples show the fanout of a port in the design. THe design comprises three inverters in a chain name *iv1*, *iv2*, and *iv3*. The *iv1* and *iv2* inverters are hierarchically combined in a larger cell named *ii2*.

To make it easier to read, we have ommitted the *container:/library/design* from the output below.

```
prompt> all_fanout -from tin
{tin ii2/hin ii2/hout iv3/in iv3/out tout}

prompt> all_fanout -from tin -flat
{tin ii2/iv1/U1/a ii2/iv1/U1/z ii2/iv2/U1/z ii2/iv2/U1/a iv3/U1/a iv3/U1/z tout}
```

SEE ALSO

```
all_fanin(2)
```

all_inputs

Creates a collection of all input ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_inputs
```

DESCRIPTION

The **all_inputs** command creates a collection of all input or inout ports in the current design.

If you want only certain ports, use the **get_ports** command to create a collection of ports matching a specific pattern and optionally passing filter criteria.

When issued from the command prompt, the **all_inputs** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change the number of objects the command displays by using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example sets constant value 0 on all input ports of the current design.

```
prompt> set_constant [all_inputs] 0
Set 'r:/WORK/top/A' to constant 0
Set 'r:/WORK/top/B' to constant 0
Set 'r:/WORK/top/C' to constant 0
```

SEE ALSO

```
collections(2)
get_ports(2)
query_objects(2)
set_constant(2)
collection_result_display_limit(3)
```

all_outputs

Creates a collection of all output ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection **all_outputs**
```

DESCRIPTION

The **all_outputs** command creates a collection of all output or inout ports in the current design.

If you want only certain ports, use **get_ports** to create a collection of ports matching a specific pattern and optionally passing filter criteria.

When issued from the command prompt, **all_outputs** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example sets all output ports of the current design as verify points.

```
prompt> set_verify_points [all_outputs]
Set verify point 'r:/WORK/bot/Z1'
Set verify point 'r:/WORK/bot/Z2'
Set verify point 'r:/WORK/bot/Z3'
```

SEE ALSO

```
collections(2)
get_ports(2)
query_objects(2)
set_verify_points(2)
collection_result_display_limit(3)
```

all_registers

Creates a collection of register cells or pins from the current design, relative to the current instance. You can assign the resulting collection to a variable or pass it into another command.

SYNTAX

```
collection all_registers
   [-cells]
   [-data_pins]
   [-clock_pins]
   [-output_pins]
   [-no_hierarchy]
```

ARGUMENTS

-cells

Creates a collection of register cells (default).

-data_pins

Creates a collection of register data pins.

-clock_pins

Creates a collection of register clock pins.

-output_pins

Creates a collection of register output pins.

-no_hierarchy

Only searches the current instance; does not descend the hierarchy.

DESCRIPTION

The **all_registers** command creates a collection of pins or cells related to registers in the current design, relative to the current instance. The collection contains objects according to the specified content control arguments (**-cells**, **-data_pins**, etc.).

When issued from the command prompt, **all_registers** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

This example gets all the register cells in the current design, descending the hierarchy.

```
prompt> current_design top
prompt> all_registers
{r:/WORK/top/U1 r:/WORK/top/U2 r:/WORK/top/M1/moltuae_reg r:/WORK/top/M1/B1/R1}
```

This example gets all data pins of all registers in the current design, not descending the hierarchy.

```
prompt> all_registers -data_pins -no_hierarchy
{r:/WORK/top/U1/Din r:/WORK/top/U1/Enable r:/WORK/top/U2/data r:/WORK/top/U2/load}
```

This example gets all nets connected to all register outputs starting from the current instance.

```
prompt> current_instance M1
prompt> get_nets -of_objects [all_registers -output_pins]
{r:/WORK/top/M1/N42 r:/WORK/top/M1/N43 r:/WORK/top/M1/B1/out}
```

SEE ALSO

```
collections(2)
current_design(2)
current_instance(2)
get_cells(2)
get_nets(2)
collection_result_display_limit(3)
```

analyze_cones

Analyze fanin cones of the specified points.

SYNTAX

```
int analyze_cones
   [-type type]
   objects\gP | -unverified | -aborted | -failing
   [-r | -i]
   [-substring substring]
   [-max_objects max_objects]
   [-reduction_factor reduction_factor]
   [-start_object start_object]
   -filter filter
   [-size size]
   [-percent percent]
   [-level level]
   [-multicone | -summary]
   [-instance instance]
   [-path]
```

Data Types

```
type string
substring string
max_points integer
reduction_factor integer
start_seed integer
filter string
size integer
percent integer
level integer
instance string
```

ENABLED SHELL MODES

Match Verify

ARGUMENTS

-type *ID_type*

Specifies the object type being analyzed:

- cell
- net
- pin
- port

objects

Analyze the specified list of objects. These can be any design objects and are not restricted to compare points. Specify the list of objects as follows:

```
{obj1 obj2 ...}
```

Use wildcard characters to match multiple objects.

-unverified

Analyzes the unverified points reported in the previous verification.

-aborted

Analyzes the aborted points reported in the previous verification.

-failing

Analyzes the failing points reported in the previous verification.

-r

Analyze the reference design objects. This option can only be used with options **-unverified**, **-aborted**, and **-failing**

-i

Analyze the implementation design objects. This option can only be used with options **-unverified**, **-aborted**, and **-failing**

-substring *substring*

Analyze only the objects containing the specified *substring*. This option can only be used with options **-unverified**, **-aborted**, and **-failing**

-max_objects *max_objects*

Stop analysis after *max_objects* objects have been analysed.

-reduction_factor *reduction_factor*

Only analyze one out of *reduction_factor* objects. This optin can be used if there are a large number of unverified, aborted, or failing points.

-start_object *start_object*

Skip the first *start_object* objects. The option **-reduction_factor** always selects the same set of objects to analyze. This option can be used to skip the first few objects and hence analyze a different set of objects.

-filter *filter*

Specify the sub-type of report being generated. Valid values of *filter* are:

- datapath
- dontcare
- logic
- pin
- powerdomain
- ungroup

Examples of each of these reports are shown in the examples section.

-size *size*

Cutoff size for instances in the fanin cone.

-percent *percent*

Cutoff percent for instances in the fanin cone. This option is similar to **-size**. The percentage is computed by comparing the instance size to the fanin cone size.

-level *level*

Cutoff level for instances in the fanin cone.

-multicone

Generate a combined report for all the specified objects. This option is only valid for filters datapath, dontcare, logic and pin.

-summary

Only prints out the cone size of specified objects. The list of objects are sorted based on the cone size.

-instance *instance*

Specifies the *instance* whose pins are to be reported. This option is only valid for filters pin.

-path

Print the complete instance path in the reports.

DESCRIPTION

This command generates statistical information about the fanin cone of objects in the design. Running this command on hard compare points can give some insight into the logic driving the point. Generating multicone reports can give an idea about the shared logic between hard compare points.

Verification can become hard because of a combination of factors. This command can help identify the following potential causes of hard verification:

- Datapath instances
- Dontcare sources
- Large XOR trees
- Un-isolated powerdomain crossings
- Ungrouped design instances and datapath instances

Many of these hard verifications can be resolved by adding cuts. One of the best places to add cuts is on pins of design instances. The fiter pin helps identify pins of design instances in the fanin cone of a given object.

EXAMPLES

The following example shows filter logic report:

```
fm_shell (match)> analyze_cones -filter logic
    -percent 20 r:/WORK/bit_coin/bit_secure_9/slice_0/piso_bit/dout_reg
**************************************************
Report         : cone_analysis
                 -filter logic
                 -percent 20
                 r:/WORK/bit_coin/bit_secure_9/slice_0/piso_bit/dout_reg

Reference      : r:/WORK/bit_coin
Implementation : i:/WORK/bit_coin
Version        : N-2017.09
Date           : Tue Jul 18 14:50:42 2017
**************************************************
Analyzing r:/WORK/bit_coin/bit_secure_9/slice_0/piso_bit/dout_reg
   Logic      XOR      SEQ     SIZE  Percent    Level  Instance (Design)
      39        0        5       44  100.00%        1  bit_coin (bit_coin)
      35        0        5       40   90.91%        2   bit_secure_9 (bit_top)
      33        0        5       38   86.36%        3    slice_0 (bit_slice)
      29        0        3       32   72.73%        4     piso_bit (piso)
      14        0        2       16   36.36%        5      dout_reg (UPF_RET_SEQ_AACC_0_0_0_1...)
**************************************************************************************************
Analysis Completed
```

The following example shows multicone filter logic report:

```
fm_shell (match)> analyze_cones -filter logic -size 12
    -multicone r:/WORK/bit_coin/bit_secure_1/slice_3*/piso_bit/dout_reg
**************************************************
Report         : cone_analysis
                 -filter logic
                 -size 12
                 -multicone
                 r:/WORK/bit_coin/bit_secure_1/slice_3*/piso_bit/dout_reg

Reference      : r:/WORK/bit_coin
Implementation : i:/WORK/bit_coin
Version        : N-2017.09
Date           : Tue Jul 18 14:53:17 2017
**************************************************
Analyzing r:/WORK/bit_coin/bit_secure_1/slice_3/piso_bit/dout_reg
Analyzing r:/WORK/bit_coin/bit_secure_1/slice_30/piso_bit/dout_reg
Analyzing r:/WORK/bit_coin/bit_secure_1/slice_31/piso_bit/dout_reg
Analyzed 3 points
 #Points  Percent    Level  Instance (Design)
       3  100.00%        1  bit_coin (bit_coin)
       3  100.00%        2    bit_secure_1 (bit_top)
       1   33.33%        3      slice_3 (bit_slice)
       1   33.33%        4        piso_bit (piso)
       1   33.33%        5          dout_reg (UPF_RET_SEQ_AACC_0_0_0_1...)
       1   33.33%        3      slice_30 (bit_slice)
       1   33.33%        4        piso_bit (piso)
       1   33.33%        5          dout_reg (UPF_RET_SEQ_AACC_0_0_0_1...)
       1   33.33%        3      slice_31 (bit_slice)
       1   33.33%        4        piso_bit (piso)
       1   33.33%        5          dout_reg (UPF_RET_SEQ_AACC_0_0_0_1...)
****************************************************************************************
Analysis Completed
```

The following example shows a summary report:

```
fm_shell (match)> analyze_cones -filter logic
   -summary r:/WORK/bit_coin/bit_secure_1/slice_3*/piso_bit/dout_reg
      44  r:/WORK/bit_coin/bit_secure_1/slice_3/piso_bit/dout_reg
      44  r:/WORK/bit_coin/bit_secure_1/slice_30/piso_bit/dout_reg
      44  r:/WORK/bit_coin/bit_secure_1/slice_31/piso_bit/dout_reg
```

SEE ALSO

`analyze_points(2)`

analyze_points

Analyzes the previous failed or incomplete verification.

SYNTAX

```
int analyze_points
   failing_aborted_or_unverified_compare_points |
   -failing | -aborted | -unverified | -all
   [-last]
   [-effort high | low]
   [-limit limit]
   [-no_operator_svp]
```

Data Types

```
failing_aborted_or_unverified_compare_points string
limit integer
```

ENABLED SHELL MODES

Verify

ARGUMENTS

failing_aborted_or_unverified_compare_points

Analyzes the specified list of failing, aborted, or unverified compare points. Specify the list of compare points as follows:

```
{cp1 cp2 ...}
```

Use wildcard characters to match multiple points.

-failing

Analyzes the failing points reported in the previous verification.

-aborted

Analyzes the aborted points reported in the previous verification.

-unverified

Analyzes the unverified points reported in the previous verification.

-all

Analyzes failing, aborted, and unverified points reported in the previous verification.

-last

Restricts the analysis to only the points considered in the previous partial verification. A verification is partial when only the compare points specified using the **set_verify_points** command are verified. If the previous verification is not a partial verification, this option has no effect.

-limit *limit*

Specifies the number of failing points to be analyzed. The default is 500. A value of *0* implies no limit.

-effort *low* | *high*

Specifies the effort level for the analysis engine. The default is *high*.

-no_operator_svp

Recommends using the Design Compiler **set_verification_priority** command with an uniquified instance. If an uniquified instance is not available, this option recommends using the command on a design, instead of datapath operators.

DESCRIPTION

This command runs a set of heuristic analyses on the previous verification to determine if there are potential causes for failing or hard verification other than logical differences. In the case of a hard verification, the **analyze_points** command might report a message that recommends using the **set_verification_priority** command in Design Compiler to obtain a verifiable result. By default, the **analyze_points** command recommends using the **set_verification_priority** command with datapath operators.

To run the **analyze_points** command automatically at the end of verification, set the **verification_run_analyze_points** variable to **true**.

To review the results of the most recent analysis, use the **report_analysis_results** command. Note that this analysis is not cumulative. The **analyze_points** command replaces all previous results.

Interrupting the **analyze_points** command stops processing but retains the partial results that are generated.

EXAMPLES

The following example shows the results of analysis on a design with missing retention behavior in the implementation design.

```
fm_shell (verify)> **analyze_points -all**
********************************** Analysis Results ***********************************
Found 1 Undriven Reference Signal
---------------------------------
An undriven signal in the reference design may be caused by either
a legitimate 'don't care' condition or an error in the RTL.
If an examination of the RTL finds no unexpected undriven signals
try 'set verification_set_undriven_signals 0:X'
to match synthesis.
------------------------------------------
r:/WORK/top/b
    Undriven in the reference cones for 2 compare point(s):
        r:/WORK/top/U0/R2_reg
        r:/WORK/top/U1/R2_reg

-------------------------------------------
**************************************************************************************
Analysis Completed
```

SEE ALSO

```
report_analysis_results(2)
set_verify_points(2)
verification_run_analyze_points(3)
```

analyze_upf

Analyze the UPF inserted into the designs using the **load_upf** command and provide diagnosis report on supply nets and power states

SYNTAX

```
status analyze_upf
   [-r]
   [-i]
   [-container container_name]
```

Data Types

```
      container_name string
```

Enabled Shell Modes

Setup
Match
Verify

ARGUMENTS

-r

Analyze the UPF information of the reference container.

-i

Analyze UPF information of the implementation container.

-container *container_name*

Analyze UPF information of the specified container.

DESCRIPTION

This command analyzes the UPF information on all containers. It provides information on the following issues in the UPF.

1. Never-On Supplies: Supplies that can never be turned on.

2. Never-true power-state constraints: Power state constraints that can never be true.

3. Mutually exclusive PSTs: Reports when power states cannot be all turned on simultaneously.

EXAMPLES

The following example reports UPF summary information on the reference container.

```
analyze_upf -r
Container: r
--------------
Found 1 Supply Net(s) that can never be turned ON
--------------
Supply Net r:/WORK/top/VDDA can never be 1 (ON value)
   Set verification_force_upf_supplies_on to false
   Use "verify -constant1 r:/WORK/top/VDDA " to see a failing logic cone for the supply net.
-----------------------

*****************************************************************************************
Analysis Completed
1
```

SEE ALSO

```
load_upf(2)
report_upf(2)
```

append_to_collection

Adds objects to a collection and modifies a variable.

SYNTAX

```
collection append_to_collection
   [-unique]
   var_name
   object_spec
```

Data Types

var_name	collection
object_spec	list

ARGUMENTS

-unique

Indicates that duplicate objects are to be removed from the resulting collection. By default, duplicate objects are not removed.

var_name

Specifies a variable name. The objects matching *object_spec* are added into the collection referenced by this variable.

object_spec

Specifies a list of named objects or collections to add.

DESCRIPTION

The **append_to_collection** command allows you to add elements to a collection. This command treats the variable name given by the *var_name* option as a collection, and it appends all of the elements in *object_spec* to that collection. If the variable does not exist, it is created as a collection with elements from the *object_spec* as its value. If the variable does exist and it does not contain a collection, it is an error.

The result of the command is the collection that was initially referenced by the *var_name* option, or the collection created if the variable did not exist.

The **append_to_collection** command provides the same semantics as the common use of the **add_to_collection** command; however, this command shows significant improvements in performance.

An example of replacing the **add_to_collection** command with the **append_to_collection** command is provided below. For example,

```
set var_name [add_to_collection $var_name $objs]
```

Using the **append_to_collection** command, the command becomes:

```
append_to_collection var_name $objs
```

The **append_to_collection** command can be much more efficient than the **add_to_collection** command if you are building up a collection in a loop. The arguments of the command have the same restrictions as the **add_to_collection** command. For more information about these restrictions, see the **add_to_collection** man page.

EXAMPLES

The following example shows how a collection can be built up using the **append_to_collection** command:

```
prompt> set xports
Error: can't read "xports": no such variable
        Use error_info for more info. (CMD-013)
prompt> append_to_collection xports [get_ports in*]
{r:/WORK/top/in0 r:/WORK/top/in1 r:/WORK/top/in2}
prompt> append_to_collection xports CLOCK
{r:/WORK/top/in0 r:/WORK/top/in1 r:/WORK/top/in2 r:/WORK/top/CLOCK}
```

SEE ALSO

```
add_to_collection(2)
foreach_in_collection(2)
index_collection(2)
remove_from_collection(2)
sizeof_collection(2)
```

apply_edits

Copies the edited designs back to the original designs.

SYNTAX

```
apply_edits
```

Enabled Shell Modes

Setup

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command copies all the edited designs created by the **edit_design** command back to the original designs.

The **verify_edits** command implicitly applies the edits. While **apply_edits** can only be used in setup mode, the **verify_edits** commands can be used in match or verify modes.

EXAMPLES

Edit a design while in verify mode, revert to setup mode, and copy the changes back:

```
fm_shell (verify)> edit_design i:/WORK/bot
```

```
    fm_shell (verify)> create_net ECO_NET_1
    fm_shell (verify)> setup
    fm_shell (setup)> apply_edits
```

SEE ALSO

```
create_net(2)
discard_edits(2)
edit_designs(2)
verify_edits(2)
```

apropos

Searches the command database for a pattern.

SYNTAX

```
string apropos
   [-symbols_only]
   pattern
```

Data Types

```
   pattern      string
```

ARGUMENTS

-symbols_only

Searches only command and option names.

pattern

Searches for the specified *pattern*.

DESCRIPTION

The **apropos** command searches the command and option database for all commands that contain the specified *pattern*. The *pattern* argument can include the wildcard characters asterisk (*) and question mark (?). The search is case-sensitive. For each command that matches the search criteria, the command help is printed as though **help -verbose** was used with the command.

Whereas **help** looks only at command names, **apropos** looks at command names, the command one-line description, option names, and option value-help strings. The search can be restricted to only command and option names with the **-symbols_only** option.

When searching for dash options, do not include the leading dash. Search only for the name.

EXAMPLES

In the following example, assume that the **get_cells** and **get_designs** commands have the **-exact** option. Note that without the **-symbols_only** option, the first search picks up commands which have the string "exact" in the one-line description.

```
prompt> apropos exact
 get_cells            # Create a collection of cells
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match cell names against patterns)

 get_designs          # Create a collection of designs
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match design names against patterns)

 real_time            # Return the exact time of day

prompt> apropos exact -symbols_only
 get_cells            # Create a collection of cells
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match cell names against patterns)

 get_designs          # Create a collection of designs
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match design names against patterns)
```

SEE ALSO

help(2)

change_link

Changes the design to which a cell is linked.

SYNTAX

```
status change_link
   cell_list
   design_name
   [ -force ]
```

Data Types

```
cell_list list

design_name string
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

RETURN VALUE

The **change_link** command returns a status of 1 if it was successful and 0 if it failed.

ARGUMENTS

cell_list

Specifies the cells for which the link is to be changed. You can specify either object IDs or instance-based paths. The specified cells must belong to the same container.

design_name

Specifies the name of the design to link to the cells specified using the *cell_list*. You can specify an objectID or just a design name. In the latter case, the libraries in the container of the *cell_list* cells are searched, and if a unique design is found, it will be used.

force

Relaxes the restriction that the new design must have extractly the same ports (same name and direction) as the pins on the cells. With this option, the new design is allowed to have extra ports.

DESCRIPTION

This command changes the design to which cells are linked. The new design must be compatible with the cells by having the same number of ports with the same name and direction as the pins on the cells.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example shows how to change the design linked to be cell b1 in the design r:/WORK/mid to r:/WORK/alt_bot.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> change_link b1 r:/WORK/alt_bot
```

If the new design name is unique across all libraries, you do not have to specify a container and a library.

```
fm_shell (setup)> current_instance i:/WORK/tile1/cpu2/core3
fm_shell (setup)> change_link cell1 AND2S4
```

SEE ALSO

```
current_design(2)
current_instance(2)
edit_design(2)
```

check_license

Checks the availability of a license for a feature.

SYNTAX

```
status check_license
   feature_list
```

Data Types

```
feature_list list
```

ARGUMENTS

feature_list

Specifies the list of features to be checked. The *feature_list* argument might consist of a single value or a space-delimited list of values enclosed within braces ({}).

By looking at your key file, you can determine all of the features licensed at your site.

RETURN VALUE

The **check_license** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command checks on a license for each of the specified features. It checks on the authorization (existence) of a license, however, it does not check out any licenses.

The list_licenses command provides a list of the features that you are currently using.

EXAMPLES

The following example checks on a Formality license:

```
fm_shell (setup)> check_license { Formality }
```

SEE ALSO

```
license_users(2)
list_licenses(2)
get_license(2)
remove_license(2)
```

collections

Describes the methodology for creating collections of objects and querying objects in the database.

DESCRIPTION

Synopsys applications build an internal database of objects and attributes applied to them. These databases consist of several classes of objects, including libraries, lib_cells, lib_pins, designs, ports, cells, pins, and nets. Most commands operate on these objects.

By definition:

A collection is a group of objects exported to the Tcl user interface.

Collections have an internal representation (the objects) and, sometimes, a string representation. The string representation is generally used only for error messages.

A set of commands to create and manipulate collections is provided as an integral part of the user interface. The collection commands encompass two categories: those that create collections of objects for use by another command, and one that queries objects for viewing. The result of a command that creates a collection is a Tcl object that can be passed along to another command. For a query command, although the visible output looks like a list of objects (a list of object names is displayed), the result is an empty string.

An empty string "" is equivalent to the empty collection, that is, a collection with zero elements.

To illustrate the usage of the common collection commands, the man pages have examples.

Homogeneous and Heterogeneous Collections

A homogeneous collection contains only one type of object. A heterogeneous collection can contain more than one type of object. Commands that accept collections as arguments can accept either type of collection.

Lifetime of a Collection

Collections are active only as long as they are referenced. Typically, a collection is referenced when a variable is set to the result of a command that creates it or when it is passed as an argument to a command or a procedure. For example, you can save a collection of design ports by setting a variable to the result of the **get_ports** command:

```
prompt> set ports [get_ports *]
```

Next, either of the following two commands deletes the collection referenced by the *ports* variable:

```
prompt> unset ports
prompt> set ports "value"
```

Collections can be implicitly deleted when they go out of scope. Collections go out of scope for various reasons. An example would be when the parent (or other antecedent) of the objects within the collection is deleted. For example, if our collection of ports is owned by a design, it is implicitly deleted when the design that owns the ports is deleted. When a collection is implicitly deleted, the variable that referenced the collection still holds a string representation of the collection. However, this value is useless because the collection is gone, as illustrated in the following example:

```
prompt> current_design
r:/WORK/top

prompt> set ports [get_ports in*]
{r:/WORK/top/in0 r:/WORK/top/in1}

prompt> remove_design top
Removing design top

prompt> query_objects $ports
Error: No such collection '_sel26' (SEL-001)
```

Iteration

To iterate over the objects in a collection, use the **foreach_in_collection** command. You cannot use the Tcl-supplied **foreach** iterator to iterate over the objects in a collection, because the **foreach** command requires a list, and a collection is not a list. In fact, if you use the **foreach** command on a collection, it destroys the collection.

The arguments of the **foreach_in_collection** command are similar to those of foreach: an iterator variable, the collection over which to iterate, and the script to apply at each iteration. Note that unlike the **foreach** command, the **foreach_in_collection** command does not accept a list of iterator variables.

The following example is an iterative way to perform a query. For more information, see the **foreach_in_collection** man page.

```
prompt> \
foreach_in_collection s1 $collection {
  echo [get_attribute $s1 name]
}
```

Manipulating Collections

A variety of commands are provided to manipulate collections. In some cases, a particular command might not operate on a collection of a specific type. This is application-specific. Consult the man pages from your application.

- **add_to_collection** - This command creates a new collection by adding a list of element names or collections to a base collection. The base collection can be the empty collection. The result is a new collection. In addition, the **add_to_collection** command allows you to remove duplicate objects from the collection by using the *-unique* option.
- **append_to_collection** - This command appends a set of objects (specified by name or collection) to an existing collection. The base collection is passed in through a variable name, and the base collection is modified directly. It is similar in function to the **add_to_collection** command, except that it modifies the collection in place; therefore, it is much faster than the

add_to_collection command when appending.

- **remove_from_collection** - This command removes a list of element names or collections from an existing collection. The second argument is the specification of the objects to remove and the first argument is the collection to have them removed from. The result of the command is a new collection. For example:

```
prompt> set dports [remove_from_collection [all_inputs] CLK]
{r:/WORK/top/in1 r:/WORK/top/in2 r:/WORK/top/in3}
```

- **compare_collections** - This command verifies that two collections contain the same objects (optionally, in the same order). The result is "0" on success.
- **copy_collection** - This command creates a new collection containing the same objects in the same order as a given collection. Not all collections can be copied.
- **index_collection** - This command extracts a single object from a collection and creates a new collection containing that object. The index operation is done in constant time - it is independent of the number of elements in the collection, or the specific index. Not all collections can be indexed.
- **sizeof_collection** - This command returns the number of objects in a collection.

Filtering

In some applications, you can filter any collection by using the **filter_collection** command. This command takes a base collection and creates a new collection that includes only those objects that match an expression.

Some applications provide a *-filter* option for their commands that create collections. This allows objects to be filtered out before they are ever included in the collection. Frequently this is more efficient than filtering after the they are included in the collection. The following examples from Formality list all techlib cells:

```
prompt> filter_collection \
[get_cells -hierarchical] {is_techlib == true}
{r:/WORK/top/U1 r:/WORK/top/M1/U2}
prompt> get_cells -hierarchical -filter {is_techlib==true}
{r:/WORK/top/U1 r:/WORK/top/M1/U2}
```

The basic form of a filter expression is a series of relations joined together with AND and OR operators. Parentheses are also supported. The basic relation contrasts an attribute name with a value through a relational operator. In the previous example, *is_hierarchical* is the attribute, == is the relational operator, and *true* is the value.

The relational operators are

```
==   Equal
!=   Not equal
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
=~   Matches pattern
!~   Does not match pattern
```

The basic relational rules are

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only true or false.

Additionally, existence relations determine if an attribute is defined or not defined, for the object. For example, in PrimeTime:

```
(sense == setup_clk_rise) and defined(sdf_cond)
```

The existence operators are

```
defined
undefined
```

These operators apply to any attribute as long as it is valid for the object class. See the appropriate man pages for complete details.

Sorting Collections

In some applications, you can sort a collection by using the **sort_collection** command. It takes a base collection and a list of attributes as sort keys. The result is a copy of the base collection sorted by the given keys. Sorting is ascending, by default, or descending when you specify the *-descending* option. In the following example, the command sorts the ports by direction and then by full name.

```
prompt> sort_collection [get_ports *] \
{direction full_name}
{r:/WORK/top/in1 r:/WORK/top/in2 r:/WORK/top/out1 r:/WORK/top/out2}
```

Implicit Query of Collections

In many applications, commands that create collections implicitly query the collection when the command is used at the command line. Consider the following examples:

```
prompt> set_constant [get_ports scan_en*] 0
Set 'r:/WORK/top/scan_en' to constant 0
Set 'r:/WORK/top/scan_en_1' to constant 0
Set 'r:/WORK/top/scan_en_2' to constant 0
prompt> get_ports in*
{r:/WORK/top/in0 r:/WORK/top/in1 r:/WORK/top/in2}
prompt> query_objects -verbose [get_ports in*]
{{port r:/WORK/top/in0} {port r:/WORK/top/in1} {port r:/WORK/top/in2}}
prompt> set inports [get_ports in*]
{r:/WORK/top/in0 r:/WORK/top/in1 r:/WORK/top/in2}
```

In the first example, the **get_ports** command creates a collection of ports that is passed to the **set_constant** command. This collection is not the result of the primary command (**set_constant**), and as soon as the primary command completes, the collection is destroyed. The second example shows how a command that creates a collection automatically queries the collection when that command is used as a primary command. The third example shows the verbose feature of the **query_objects** command, which is not available with an implicit query. Finally, the fourth example sets the variable **inports** to the result of the **get_ports** command. Only in the final example does the collection persist to future commands until **inports** is overwritten, unset, or goes out of scope.

SEE ALSO

```
add_to_collection(2)
append_to_collection(2)
compare_collections(2)
copy_collection(2)
filter_collection(2)
foreach_in_collection(2)
get_attribute(2)
index_collection(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
sort_collection(2)
```

commit_edits

Commits the changes made by edit commands in the current session.

SYNTAX

```
status **commit_edits**
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command commits the changes made to the circuit by edit commands. It also removes the backup libraries containing the unedited designs.

A design is edited when you run the edit commands. Committing the edits does not change the functionality of the circuit. However, committing the edits sets a check-point for future **undo_edits** commands.

By default, the tool commits edits before performing operations that make changes to the circuit. An example of this is processing SVF.

Before committing the edits, you can review them using the **compare_edits** command. You can also highlight the edits in the GUI by selecting the **Color Edits** option in the **ECO** menu. For more information about editing a design, see the *Formality User Guide*.

EXAMPLES

The following example creates a new net N1 in design r:/WORK/mid, commits the edit, creates a new N2, and reverts the change again.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_net N1
fm_shell (setup)> commit_edits
fm_shell (setup)> create_net N2
fm_shell (setup)> undo_edits
```

SEE ALSO

```
create_net(2)
current_design(2)
edit_design(2)
undo_edits(2)
```

compare_collections

Compares the contents of two collections. If the same objects are in both collections, the result is "0" (like string compare). If they are different, the result is nonzero. The order of the objects can optionally be considered.

SYNTAX

```
int compare_collections
   [-order_dependent]
   collection1
   collection2
```

Data Types

collection1	collection
collection2	collection

ARGUMENTS

-order_dependent

Indicates that the order of the objects is to be considered; that is, the collections are considered to be different if the objects are ordered differently.

collection1

Specifies the base collection for the comparison. The empty string (the empty collection) is a legal value for the *collection1* argument.

collection2

Specifies the collection with which to compare to *collection1*. The empty string (the empty collection) is a legal value for the *collection2* argument.

DESCRIPTION

The **compare_collections** command is used to compare the contents of two collections. By default, the order of the objects does not matter, so that a collection of cells u1 and u2 is the same as a collection of the cells u2 and u1. By using the **-order_dependent** option, the order of the objects is considered.

Either or both of the collections can be the empty string (the empty collection). If two empty collections are compared, the comparison succeeds (that is, **compare_collections** considers them identical), and the result is "0".

EXAMPLES

The following example shows a variety of comparisons. Note that a result of "0" from **compare_collections** indicates success. Any other result indicates failure.

```
prompt> compare_collections [get_cells *] [get_cells *]
0
prompt> set c1 [get_cells {u1 u2}]
{r:/WORK/top/u1 r:/WORK/top/u2}
prompt> set c2 [get_cells {u2 u1}]
{r:/WORK/top/u2 r:/WORK/top/u1}
prompt> set c3 [get_cells {u2 u4 u6}]
{r:/WORK/top/u2 r:/WORK/top/u4 r:/WORK/top/u6}
prompt> compare_collections $c1 $c2
0
prompt> compare_collections $c1 $c2 -order_dependent
-1
prompt> compare_collections $c1 $c3
-1
```

The following example builds on the previous example by showing how empty collections are compared.

```
prompt> set c4 ""
prompt> compare_collections $c1 $c4
-1
prompt> compare_collections $c4 $c4
0
```

SEE ALSO

```
collections(2)
```

compare_edits

Identifies the edits to a design and compares them with the backup copy of the design.

SYNTAX

```
status compare_edits
   [ -list ]
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-list

Reports the edits as a Tcl list. You can use this list in further Tcl processing.

RETURN VALUE

By default, the **compare_edits** command returns a status of 1.

When you specify the *-list* option, the command returns a Tcl list. At the outermost level, the list contains alternating keys and values. The keys are the strings "added", "removed", and "changed". The corresponding values are lists encoding the added, removed, and changed objects. These value are lists on the form of alternating keys and values. Here the keys are element types ("pin", "port", "net", and "cell"), and the associated values are lists of elements of those types.

When the lists of elements are empty, the following is the Tcl list that the command returns:

```
added   {pin { } port { } net { } cell { }}
removed {pin { } port { } net { } cell { }}
changed {pin { } port { } net { } cell { }}
```

DESCRIPTION

This command is used to determine the parts of a netlist that are affected by edit commands such as **create_net** and **disconnect_net**.

The **compare_edits** command considers uncommitted edits; edits that are not yet committed using \ the **commit_edits** command.

The **compare_edits** command reports the pins, ports, nets, and cells that are either added, removed, or changed. Added elements are reported as objects in the design libraries (normally WORK). Removed elements are reported as objects in the backup libraries (normally FM_BACKUP_WORK). Changed elements are reported as objects from both types of libraries.

You can also use the **compare_edits** functionality in the GUI from the **ECO** menu.

EXAMPLES

The following example shows how to use the **compare_edits** command.

```
fm_shell (setup)> compare_edits
ADDED elements:
        NETS
                i:/WORK/bot/new


REMOVED elements:
        PINS
                i:/FM_BACKUP_WORK/mid/b1/bo1
                i:/FM_BACKUP_WORK/mid/b2/bo1

        PORTS
                i:/FM_BACKUP_WORK/bot/bo1


CHANGED elements:
        NETS
                i:/FM_BACKUP_WORK/bot/bo1
                i:/FM_BACKUP_WORK/mid/mo1
                i:/FM_BACKUP_WORK/mid/mo2
                i:/WORK/bot/bo1
                i:/WORK/mid/mo1
                i:/WORK/mid/mo2

        CELLS
                i:/FM_BACKUP_WORK/mid/b1
                i:/FM_BACKUP_WORK/mid/b2
                i:/WORK/mid/b1
                i:/WORK/mid/b2
```

When you use the *-list* option, the are reported in a Tcl list.

```
fm_shell (setup)> compare_edits -list
added   {pin  { } port { } net { i:/WORK/bot/new } cell { }}
removed {pin  { i:/FM_BACKUP_WORK/mid/b1/bo1 i:/FM_BACKUP_WORK/mid/b2/bo1 }
         port { i:/FM_BACKUP_WORK/bot/bo1 } net { } cell { }}
changed {pin  { } port { }
         net  { i:/FM_BACKUP_WORK/bot/bo1 i:/FM_BACKUP_WORK/mid/mo1 i:/FM_BACKUP_WORK/mid/mo2
                i:/WORK/bot/bo1 i:/WORK/mid/mo1 i:/WORK/mid/mo2 }
         cell { i:/FM_BACKUP_WORK/mid/b1 i:/FM_BACKUP_WORK/mid/b2 i:/WORK/mid/b1 i:/WORK/mid/b2 }}
```

SEE ALSO

```
create_net(2)
disconnect_net(2)
commit_edits(2)
```

connect_net

Connects a net to one or more pins.

SYNTAX

```
status connect_net
   net_name
   pin_list
```

Data Types

```
net_name string
pin_list list
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

net_name

Specifies the name of the net to connect. You can specify either an object ID or an instance-based path.

The command connects nets in a design and the changes are visible to all instances of that design. To connect a specific instance-based path, uniquify the instance of the design before connecting it.

pin_list

Specifies the pins to which the net is to be connected. You can specify either object IDs or instance-based paths.

RETURN VALUE

The **connect_net** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command connects a net to the specified pins. If the net is not in the same hierarchy level as the pins, the tool punches ports and creates new net segments to enable the connection.

You can connect a net to many pins. However, you cannot connect a pin to more than one net. If you connect a net to a pin that already has an existing net connection, the existing net is disconnected.

To disconnect pins from a net, use the **disconnect_net** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example shows how to connect net net3 to two pins, r:/WORK/top/m1/i1 and r:/WORK/top/m1/b1/bi1.

```
fm_shell (setup)> current_instance r:/WORK/top/m1
fm_shell (setup)> connect_net net3 {i1 b1/bi1}
```

SEE ALSO

```
current_design(2)
current_instance(2)
disconnect_net(2)
edit_design(2)
```

connect_pin

Connects pins or ports at any level of hierarchy.

SYNTAX

```
status connect_pin
   -from from_object
   -to to_list
```

Data Types

```
from_object string
to_list list
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-from from_object

Specifies the pin or port from which to make the connection. You can specify either an object ID or an instance-based path.

-to to_list

Specifies the pins and ports to which to make the connection. The pins and ports can be at any level of the hierarchy. You can specify either object IDs or instance-based paths.

RETURN VALUE

The **connect_pin** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command performs global connections between the source object specified in the **-from** option and the objects specified in the **-to** option. The specified pins and ports can be at any level of hierarchy.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example shows how to connect pin U1/Z to two pins, r:/WORK/top/m1/i1 and r:/WORK/top/m1/b1/bi1.

```
fm_shell (setup)> current_instance r:/WORK/top/m1
fm_shell (setup)> connect_pin -from U1/Z -to {i1 b1/bi1}
```

SEE ALSO

```
current_design(2)
current_instance(2)
connect_net(2)
edit_design(2)
```

copy_collection

Duplicates the contents of a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection copy_collection
   collection1
```

Data Types

collection1	collection

ARGUMENTS

collection1

Specifies the collection to be copied. If an empty string is used for the *collection1* argument, the command returns the empty string (a copy of the empty collection is an empty collection).

DESCRIPTION

The **copy_collection** command is an efficient mechanism for creating a duplicate of an existing collection. It is more efficient and almost always sufficient to simply have more than one variable referencing the same collection. For example, if you create a collection of ports and save a reference to it in a **c1** variable, assigning the value of the **c1** variable to another **c2** variable creates a second reference to the same collection:

```
prompt> set c1 [get_cells U1*]
{r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U11 r:/WORK/top/U12}
prompt> set c2 $c1
{r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U11 r:/WORK/top/U12}
```

This has not copied the collection. There are now two references to the same collection. If you change the *c1* variable, the *c2* variable continues to reference the same collection:

```
prompt> set c1 [get_cells block1]
{r:/WORK/top/block1}
prompt> query_objects $c2
{r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U11 r:/WORK/top/U12}
```

There might be instances when you really do need a copy. In those cases, the **copy_collection** command is used to create a new collection that is a duplicate of the original.

EXAMPLES

The following example shows the result of copying a collection. Functionally, it is not much different that having multiple references to the same collection.

```
prompt> set c1 [get_cells U1*]
{r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U11 r:/WORK/top/U12}
prompt> set c2 [copy_collection $c1]
{r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U11 r:/WORK/top/U12}
prompt> unset c1
prompt> query_objects $c2
{r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U11 r:/WORK/top/U12}
```

SEE ALSO

```
collections(2)
```

cputime

Returns the CPU time used by the tool's shell.

SYNTAX

```
cputime
```

ARGUMENTS

none

DESCRIPTION

This command returns the CPU time used by the tool's shell. The time is rounded off to the nearest one hundredth of a second.

EXAMPLES

The following example shows the output produced by the **cputime** command.

```
fm_shell (setup)> cputime
3.73
fm_shell (setup)>
```

create_cell

Creates new cells.

SYNTAX

```
cell_list create_cell
   [cell_list]
   reference_name
   [ -connections pin_connection_list ]
```

Data Types

```
      cell_list list
      reference_name string
      pin_connection_list list
```

RETURN VALUE

The **create_cell** command returns a list of the cells it created.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

cell_list

Specifies the names of cells to create. You can specify either object IDs or instance-based paths. Each cell name must be unique within the design.

Cells are created in designs and are visible to all instances of that design. If you want to only affect a particular instance-based path, you must uniquify that instance of the design first.

If you do not specify this argument, the tool creates a single new cell and generates a name. The name is of the form *<prefix>_CELL_<number>*, where <prefix> is controlled by the **current_prefix** command, and <number> is an integer to make the name unique.

Use the *cell_list* argument before using the *reference_name* argument.

reference_name

Specifies the design or library cell that the new cells reference. Ports on the reference determine the name, number, and direction of pins on the new cell.

pin_connection_list

Specifies the pin connections for the new cell as a list of "<pin>=<net>" elements. Note that there are no spaces before or after the equal sign.

When the nets are in a different level of hierarchy than the newly created cell, the command creates ports and extra nets to make the connection.

DESCRIPTION

This command creates new cells based on the *cell_list* argument. New cells are the instantiation of an existing design or library cell.

To remove cells from the current design, use the **remove_cell** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example creates two new cells in design r:/WORK/mid. The new cells are called *box1* and *box2*, and they are both instantiations of design r:/WORK/bot.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_cell {box1 box2} r:/WORK/bot
```

If the design name is unique across all libraries, you do not have to specify a container and a library:

```
fm_shell (setup)> current_design i:/WORK/cpu
fm_shell (setup)> create_cell {cell1 cell2} AND2S4
```

You can use instance-based paths to create a cell and connect it up to four nets. Note that the four nets are specified relative to the the current instance (pins A and B), via the return value of a call to **create_net** (pin C), or by a full instance-base path (pin Z):

```
fm_shell (setup)> current_instance i:/WORK/top/m1/b1
fm_shell (setup)> create_cell AND3
  -connections [list A=../net1 B=cell3/net2 C=[create_net] Z=i:/WORK/top/m2/net3]
```

SEE ALSO

```
create_net(2)
current_design(2)
current_instance(2)
current_prefix(2)
edit_design(2)
remove_cell(2)
```

create_command_group

Creates a new command group.

SYNTAX

```
string create_command_group [-info info_text]
group_name
```

ARGUMENTS

-info *info_text*

Help string for the group

group_name

Specifies the name of the new group.

DESCRIPTION

The **create_command_group** command is used to create a new command group, which you can use to separate related user-defined procedures into functional units for the online help facility. When a procedure is created, it is placed in the "Procedures" command group. With the **define_proc_attributes** command, you can move the procedure into the group you created.

The *group_name* can contain any characters, including spaces, as long as it is appropriately quoted. If *group_name* already exists, **create_command_group** quietly ignores the command. The result of **create_command_group** is always an empty string.

EXAMPLES

The following example demonstrates the use of the **create_command_group** command:

```
prompt> create_command_group {My Procedures} -info "Useful utilities"

prompt> proc plus {a b} { return [expr $a + $b] }

prompt> define_proc_attributes plus -command_group "My Procedures"

prompt> help
My Procedures:
  plus

 ...
```

SEE ALSO

```
define_proc_attributes(2)
help(2)
proc(2)
```

create_constraint_type

Generates a user-defined constraint type.

SYNTAX

```
status create_constraint_type
   type_name
   designID
```

Data Types

```
type_name string
designID string
```

ARGUMENTS

type_name

Specifies the name of the constraint type.

designID

Specifies the name of the design that defines the constraint.

DESCRIPTION

This command creates a user-defined constraint type from the specified design. You can then use the constraint type can with the **set_constraint** command to constrain control points in the verification. For more information, see the **set_constraint** command man page.

The following rules apply when you specify the constraint module using the *designID* argument:

- One or more inputs
- Only one output

- Output state is logic 1 if the inputs to the module satisfy the constraint (the control points are in a legal state). Output state is logic 0 if the inputs to the module do not satisfy the constraint (the control points are not in a legal state).
- No inouts
- No sequential logic
- No cycles
- No three-state logic
- No black boxes

The *type_name* argument defines the name used to reference this constraint type using the **set_constraint** command. The *type_name* argument must be unique from the set of predefined and user-defined type names.

The constraint module is automatically elaborated when you run the **create_constraint_type** command.

EXAMPLES

The following example reads a design that is used to create a user-defined constraint type. The constraint module file my_2hot.v allows only two inputs to be 1 at the same time.

```
module my_2hot(in1, in2, in3 out);
input  in1, in2, in3;
output out;
reg out;

always @*
case ({in1, in2, in3})
3'b110 : out = 1'b1;
3'b011 : out = 1'b1;
3'b101 : out = 1'b1;
default: out = 1'b0;
endcase

endmodule

fm_shell> read_verilog -container ctype my_2hot.v
No target library specified, default is WORK
Loading verilog file 'my_2hot.v'
Created container 'ctype'
Current container set to 'ctype'
1
fm_shell> create_constraint_type 2hot ctype:/WORK/my_2hot
1
```

SEE ALSO

```
    remove_constraint(2)
    remove_constraint_type(2)
    report_constraint(2)
    report_constraint_type(2)
    set_constraint(2)
```

create_container

Creates a new container.

SYNTAX

```
create_container
   containerID
```

Data Types

```
containerID string
```

ARGUMENTS

containerID

Specifies the name of the container to create.

DESCRIPTION

This command creates an empty container and establishes it as the current container. If a container with the specified name already exists, the tool reports an error.

When you create a container, the command loads the GTECH technology library, and any other shared technology libraries, into the new container.

For more information about containers, see the *Formality User Guide*.

The **create_container** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

This example creates a new container, ref, and establishes it as the current container.

```
fm_shell> create_container ref
Created container 'ref'
Current container set to 'ref'
1
fm_shell>
```

SEE ALSO

```
current_container(2)
report_containers(2)
```

create_cutpoint_blackbox

Creates a single-input, single-output black box.

SYNTAX

```
create_cutpoint_blackbox
   blackbox_name
   objectID
   [-type objectID_type]
   [-invert]
```

Data Types

```
   blackbox_name string
   objectID string
   objectID_type string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

blackbox_name

Specifies the name of the black box to insert.

objectID

Specifies the design object into which the command inserts the black box.

-type *objectID_type*

Specifies the type of the object that is specified by the *objectID* argument. Use this option if the name of the specified design object is associated with more than one type of design object within the same design. Specify one of the following for the *objectID_type* argument:

- *pin* for a pin type
- *net* for a net type

-invert

Specifies that the inserted cutpoint is inversed.

DESCRIPTION

This command inserts a single-input, single-output black box on the specified net or the net of the specified pin.

The black box input pin is named *In* and the output pin is named *Out*. The inserted black box is a cutpoint that effectively breaks the net on which it is inserted. If you specify a net object, then the net is disconnected from its drivers and connected to the black box output pin instead. A new net is then created that connects all drivers of the specified net to the black box input pin.

Similarly, the net that drives a specified pin is cut. The existing net connects to the black box input pin, and a new net is generated to connect the black box output pin to the specified pin.

This command does not work with bidirectional pins or nets connected to bidirectional pins. This command forces the creation of a compare point and inserts a new primary input at the specified net or the net of the specified pin, because the tool creates compare points at black box input pins by default.

EXAMPLES

```
fm_shell (setup)> create_cut myblackbox ref:/WORK/scanner/updn -type net
Created cutpoint black box 'ref:/WORK/scanner/myblackbox' with input pin 'In' and output pin 'Out'.
Net connected to input pin: 'ref:/WORK/scanner/N0'
Net connected to output pin: 'ref:/WORK/scanner/updn'
1
fm_shell (setup)>
```

SEE ALSO

```
set_cutpoint(2)
remove_cutpoint(2)
report_cutpoints(2)
```

create_eco_patch

Creates an ECO change list (ECO patch).

SYNTAX

```
status create_eco_patch
   [ -prefix name ]
   [ -replace ]
   [ filename ]
```

Data Types

```
name string
filename string
```

Enabled Shell Modes

Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

ARGUMENTS

-prefix *name*

Specifies a string that will be used as a name prefix for all new nets/cells/ports. The default name prefix is "ECO_".

-replace

Overwrites output file if it already exists.

filename

Specifies the name of the output file. If not specified, the default filename "fm_eco_region.patch.tcl" is used.

DESCRIPTION

This command can only be issued in match or verify mode. You must first specify the original implementation, the ECO reference, the ECO implementation and source the ECO regions data file that was generated during a match_eco_regions session. This command will then create the netlist edit commands that when applied to the original implementation will make it functionally equivalent to the ECO reference.

EXAMPLES

The following example creates an ECO patch.

```
fm_shell (match)> create_eco_patch -replace
```

SEE ALSO

```
match_eco_regions
write_eco_regions
set_orig_reference
set_orig_implementation
set_eco_reference
set_eco_implementation
```

create_net

Creates new nets.

SYNTAX

```
net_list create_net
   [ net_list ]
   [ -power | -ground | -tie_high | -tie_low ]
   [ -pins pin_list ]
```

Data Types

```
net_list list
pin_list list
```

RETURN VALUE

The **create_net** command returns a list of the nets it created.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

net_list

Specifies the names of the nets to create. You can specify either object IDs or instance-based paths. Each net name must be unique in the design.

Nets are created in designs and are visible to all instances of that design. If you want to only affect a particular instance-based path, you must uniquify the instance of the design first.

If you do not specify this argument, the tool creates a single new net and generates a name. The name is of the form *<prefix>_NET_<number>*, where <prefix> is controlled by the **current_prefix** command, and <number> is an integer to make the name unique.

-power

Creates a power net. By default, the tool creates a signal net.

The *-power*, *-ground*, *-tie_high*, and *-tie_low* options are mutually exclusive.

-ground

Creates a ground net. By default, the tool creates a signal net.

The *-power*, *-ground*, *-tie_high*, and *-tie_low* options are mutually exclusive.

-tie_high

Creates a constant 1 net. By default, the tool creates a signal net.

The *-power*, *-ground*, *-tie_high*, and *-tie_low* options are mutually exclusive.

-tie_low

Creates a constant 0 net. By default, the tool creates a signal net.

The *-power*, *-ground*, *-tie_high*, and *-tie_low* options are mutually exclusive.

pin_list

Specifies the pins to which the new net is connected. This option is only allowed when creating a single net.

When the pins are in a different level of hierarchy than the newly created net, the command creates ports and extra nets to make the connection.

Pins that already are connected to a net are disconnected before they are connected to the newly created net.

DESCRIPTION

This command creates new net objects based on the *net_list* argument. The **create_net** command creates only scalar (single bit) nets.

Use the *-pins* option to connect nets to pins, or use the **connect_net** command. To remove nets, use the **remove_net** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

Logically, the *-ground* and *-tie_low* options, and the *-power* and *-tie_high* options are equivalent.

EXAMPLES

The following example shows how to create two new nets in design *r:/WORK/mid*. The new cells are named *net1* and *net2*.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_net {net1 net2}
```

Assuming *r:/WORK/top/m1* is an instance of *r:/WORK/mid*, use instance-based paths to create a new net in *r:/WORK/mid*:

```
fm_shell (setup)> current_instance r:/WORK/top/m1
fm_shell (setup)> create_net net3 -pins {../m2/IN1 IN2 and_gate_cell/Z}
```

SEE ALSO

```
connect_net(2)
create_cell(2)
current_design(2)
current_instance(2)
current_prefix(2)
disconnect_net(2)
edit_design(2)
remove_net(2)
```

create_port

Creates ports.

SYNTAX

```
port_list create_port
   [ port_list ]
   [ -direction dir ]
```

Data Types

```
      port_list list
      dir string
```

RETURN VALUE

The **create_port** command returns a list of the ports it created.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

port_list

Specifies the names of ports to create. You can use either object IDs or instance-based paths. Each port name must be unique within the current design.

Ports are created in designs and are visible to all instances of that design. If you want to only affect a particular instance-based path, you must uniquify that instance of the design first.

If you do not specify this argument, the tool creates a single new port and generates a name. The name is of the form *<prefix>_PORT_<number>*, where <prefix> is controlled by the **current_prefix** command, and <number> is an integer to make the name unique.

-direction dir

Specifies the signal flow of the created port. Specify one of the following:

- **in** - Default.
- **out**
- **inout**

DESCRIPTION

This command creates new port objects based on the *port_list* argument.

Ports are the external connection points on a design. To connect ports to nets inside a design, use the **connect_net** or **create_net** commands.

Use the **remove_port** command to remove ports.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example shows how to create two new ports on design *r:/WORK/mid*. The new ports are named *new_in* and *new_out*.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_port new_in -dir in
fm_shell (setup)> create_port new_out -dir out
```

This example shows how to create a new port and connect it to a new net:

```
fm_shell (setup)> current_instance r:/WORK/top/m1/b1
fm_shell (setup)> create_net -pins [create_port]
```

SEE ALSO

```
connect_net(2)
create_cell(2)
current_design(2)
current_instance(2)
current_prefix(2)
edit_design(2)
remove_port(2)
```

create_primitive

Creates primitive cells.

SYNTAX

```
cell_list create_primitive
   [cell_list]
   cell_type
   [ -size size_val ]
   [ -connections pin_connection_list ]
```

Data Types

```
cell_list list
cell_type string
size_val integer
pin_connection_list list
```

RETURN VALUE

The **create_primitive** command returns a list of the cells it created.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

cell_list

Specifies the names of cells to create. You can use either object IDs or instance-based paths. Each cell name must be unique within its enclosing design.

Cells are created in designs and are visible to all instances of the design. To affect a specific instance-based path, uniquify the instance of the design first.

If you do not specify this argument, the tool creates a single new cell and generates a name. The name is of the form *<prefix>_CELL_<number>*, where <prefix> is controlled by the **current_prefix** command, and <number> is an integer to make the name unique.

Use the *cell_list* argument before using the *cell_type* argument.

cell_type

Specifies the type of primitive cells to create. The type must be one of the following:

- AND - to create combinational logic gate cells
- OR - to create combinational logic gate cells
- XOR - to create combinational logic gate cells
- NAND - to create combinational logic gate cells
- NOR - to create combinational logic gate cells
- XNOR - to create combinational logic gate cells
- BUF - to create buffer cells
- INV - to create inverter cells
- DC - to create dont-care cells
- TRI - to create tri-state cells
- SEQ - to create register cells

size_val

Specifies the number of input pins of the gates being created. The size argument is ignored for BUF, INV, DC, TRI, and SEQ cell types. The size must be 2 or larger for XOR and XNOR. The size must be 0 or larger for AND, OR, NAND, and NOR. The default is 2.

pin_connection_list

Specifies the pin connections for the new cell as a list of "<pin>=<net>" elements. Note that there are no spaces before or after the equal sign.

When the nets are in a different level of hierarchy than the newly created cell, the command creates ports and extra nets to make the connection.

DESCRIPTION

This command creates new primitive cells.

Use the **invert_pin** command to invert pins on primitive cells.

The output pins on non-SEQ primitive cells are always called *OUT*. SEQs have two output pins: *Q* and *QN*.

The input pins on combinational logic primitive gates are called *IN1*, *IN2*, etc.

The input pins on dont_care cells are called *DC* and *F*.

The input pins on TRI cells are called *D* and *EN*.

The input pins on SEQ cells are called *CLK*, *SL*, *SD*, *SS*, *SC*, *ST*, *AL*, *AD*, *AS*, and *AC*.

To create cells that are instantiations of a design, use the **create_cell** command.

To remove cells from the current design, use the **remove_cell** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

This command is specific to Formality. It is not supported by other Synopsys tools. Formality will warn about this the first time the command is used.

EXAMPLES

The following example creates a new 3-input AND cell in the design r:/WORK/mid. The new cell is named *MY_AND_GATE*.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_primitive MY_AND_GATE AND -size 3
```

This example creates a new 2-input XOR cell in the design of instance r:/WORK/top/m1/b1. The new cell is called *MY_XOR_GATE*.

```
fm_shell (setup)> current_instance r:/WORK/top/m1/b1
fm_shell (setup)> create_primitive MY_XOR_GATE XOR
```

This is the previous example without the use of **current_instance**.

```
fm_shell (setup)> create_primitive r:/WORK/top/m1/b1/MY_XOR_GATE XOR
```

This example creates a new 3-input AND cell in the design of instance r:/WORK/top/m1. The new cell is called *MY_AND_GATE*. Pins IN1, IN2, and IN3 of *MY_AND_GATE* are connected to nets m1 of mi2, i3 of top, and i4 of top, respectively. Since no absolute instance paths were used, all nets are referenced relative to the **current_instance** top.

```
fm_shell (setup)> current_instance r:/WORK/top
fm_shell (setup)> create_primitive m1/MY_AND_GATE and -size 3
     -connect {IN1=m1/mi2 IN2=i3 IN3=i4 OUT=m1/b1/bo2}
```

This example uses auto-naming to create a new SEQ in the **current_instance** top.

```
fm_shell (setup)> current_instance r:/WORK/top
fm_shell (setup)> set name [create_primitive seq]
fm_shell (setup)> puts "Name: $name."
Name: r:/WORK/top/FM_CELL_1.
```

SEE ALSO

```
create_cell(2)
current_design(2)
current_instance(2)
edit_design(2)
invert_pin(2)
remove_cell(2)
```

current_container

Sets or reports the current container.

SYNTAX

```
status current_container
   [ containerID ]
```

Data Types

```
containerID string
```

ARGUMENTS

containerID

Specifies the name of the container to establish as the current container.

DESCRIPTION

This command establishes or reports the current container.

If the specified container ID does not exist, the tool reports the following error:

```
Error: container name could not be found.
```

To report the current container, enter the command without specifying a container ID. If the specified container is not set, the tool reports the following error:

```
The current container is not set
0
```

The current container remains current until you issue a subsequent **current_container** command or create a new container. For conceptual information about containers, see the *Formality User Guide*.

The **current_container** command returns one of the following:

- 0 to indicate failure
- A string containing the name of the current container

EXAMPLES

The following example creates three containers ref, impl, and temp, establishes the container named temp as the current container, and then reports the current container.

```
fm_shell> create_container ref
Created container 'ref'
Current container set to 'ref'
1
fm_shell> create_container impl
Created container 'impl'
Current container set to 'impl'
1
fm_shell> create_container temp
Created container 'temp'
Current container set to 'temp'
1
fm_shell> current_container temp
impl
fm_shell> current_container
impl
fm_shell>
```

SEE ALSO

```
create_container(2)
report_containers(2)
```

current_design

Sets or reports the current design.

SYNTAX

```
current_design
   [ designID ]
```

Data Types

```
designID string
```

ARGUMENTS

designID

Specifies the name of the design to establish as the current design. For information about design IDs, see the *Formality User Guide*.

DESCRIPTION

This command sets or reports the current design.

To establish a current design, specify the *designID* argument. If the specified design does not exist, the tool reports one of the following two error messages:

```
Error: DesignID name is invalid
Error: A design named name could not be found in container
```

Note: Reading in a design does not establish it as the current design.

To report the current design, run the command without specifying the *designID* argument. If a current design is not set, the tool reports the following error:

```
The current design is not set
0
```

The current design remains current until you issue the subsequent **current_design** command.

EXAMPLES

The following example establishes and reports the current design.

```
fm_shell> create_container ref
Created container 'ref'
Current container set to 'ref'
1
fm_shell> read_db mapped_gate_lca500k.db
Loading db file 'mapped_gate_lca500k.db'
No target library specified, default is WORK
1
fm_shell> current_design /WORK/CORE
ref:/WORK/CORE
fm_shell> current_design
ref:/WORK/CORE
fm_shell>
```

SEE ALSO

`current_instance(2)`

current_instance

Sets or reports the current instance.

SYNTAX

```
current_instance
   [ instancePath ]
```

Data Types

```
instancePath string
```

RETURN VALUE

The **current_instance** command returns the absolute path of the current instance.

ARGUMENTS

instancePath

Specifies the working cell.

DESCRIPTION

The **current_instance** command sets the working instance, and enables you to traverse the design hierarchy. The instance specified by the **current_instance** command is a cell embedded in the hierarchy of a design. You can use the current instance as the hierarchical reference point in many Formality commands to specify the hierarchy in which design objects such as nets, pins, and ports are found.

The **current_instance** command always operates relative to the value of the **current_design** command. It may be identical to the **current_design** command, or below the **current_design** in the hierarchy. If the current instance is not set, the command assumes the value of the **current_design** command.

The **current_instance** command can take either no argument, or an instance path:

- If no argument is specified, it returns the value of the **current_design** command.
- If the **current_instance** command is set to an absolute or fully qualified instance path, the specified cell becomes the current cell, and the **current_design** command is set to the top-level design of the path.
- If the **current_instance** command is set to a relative instance path, the specified cell becomes the current cell, and only the **current_instance** command is affected; the **current_instance** command is defined relative to the **current_design** command.

An instance path is relative if it does not fully specify the path (container, library, instance path) to the cell:

- A "." signifies current directory.
- A ".." moves one level up the design hierarchy. Note that ".." directive may also be either nested or embedded, or both, in complex instance instantations, as shown in the following examples.
- If a valid cell at the current level of hierarchy is specified, the **current_instance** command is moved down to that level of hierarchy.

The current instance remains current until either the subsequent **current_instance** or **current_design** command.

EXAMPLES

The following examples shows the interplay between **current_instance** and **current_design**.

```
fm_shell> current_instance
The current instance is not set
0
fm_shell> current_design
The current design is not set
0
fm_shell> current_instance r:/WORK/top
r:/WORK/top

## no args echoes current_instance

fm_shell> current_instance
r:/WORK/top

fm_shell> current_design
r:/WORK/top

## invalid current_instance un-sets current_design

fm_shell> current_instance ../
```

```
Error: Unknown name: '../' (FM-036)
Information: Current design is no longer set (FM-072)
0
fm_shell> current_instance
The current instance is not set
0
## Setting current_instance full path sets current_design to top

fm_shell> current_instance r:/WORK/top/m1
r:/WORK/top/m1

fm_shell> current_design
r:/WORK/top

## Relative cell reference

fm_shell> current_instance b1
r:/WORK/top/m1/b1

## Another relative cell reference

fm_shell> current_instance ../b2
r:/WORK/top/m1/b2

## Re-setting current_design un-sets current_instance

fm_shell> current_design r:/WORK/mid
r:/WORK/mid
fm_shell> current_instance
The current instance is not set
0
fm_shell>
```

SEE ALSO

```
current_prefix(2)
current_design(2)
create_net(2)
create_port(2)
create_cell(2)
connect_net(2)
remove_net(2)
remove_port(2)
remove_cell(2)
change_link(2)
record_edits(2)
undo_edits(2)
write_edits(2)
```

current_prefix

Sets or reports the prefix used by automatic name generation in high-level ECO editing.

SYNTAX

```
current_prefix
   [ prefix ]
```

Data Types

```
prefix string
```

RETURN VALUE

The **current_prefix** command returns the name of the current prefix.

ARGUMENTS

prefix

Appends the specified prefix to design objects created using the auto-name-generation feature of the high level editing ECO command set.

DESCRIPTION

The **current_prefix** command appends the specified prefix to objects created by the automatic name-generation feature of the following high level editing ECO commands:

```
create_net
create_port
```

```
    create_cell
```

The auto-naming format for the three commands is <prefix>_<type>_<integer>, where:

```
    <prefix> is current_prefix
    <type> is NET, PORT OR CELL, respectively
    <integer> is the lowest available integer to guarantee uniqueness of the name
```

The default of **current_prefix** is "FM" .

EXAMPLES

The following example creates a net using the default value of **current_prefix** and connects it to the specified pin.

```
  fm_shell (setup)> create_net -pins ../register/U133/zn
  i:/WORK/mR4000/cntrl/FM_NET_1
```

The following example overrides the default value of **current_prefix** and connects it to a given pin.

```
  fm_shell (setup)> current_prefix ECO7
  fm_shell (setup)> create_net -pins ../register/U131/zn
  i:/WORK/mR4000/cntrl/ECO7_NET_1
```

SEE ALSO

```
current_instance(2)
create_net(2)
create_port(2)
create_cell(2)
```

date

Returns a string containing the current date and time.

SYNTAX

```
string date
```

DESCRIPTION

The **date** command generates a string containing the current date and time, and returns that string as the result of the command. The format is fixed as follows:

```
ddd mmm nn hh:mm:ss yyyy
```

Where:

```
ddd is an abbreviation for the day
mmm is an abbreviation for the month
nn is the day number
hh is the hour number (24 hour system)
mm is the minute number
ss is the second number
yyyy is the year
```

The **date** command is useful because it is native. It does not fork a process. On some operating systems, when the process becomes large, no further processes can be forked from it. With it, there is no need to call the operating system with **exec** to ask for the date and time.

EXAMPLES

The following command prints the date.

```
prompt> echo "Date and time: [date]"
Date and time: Thu Dec  9 17:29:51 1999
```

SEE ALSO

exec(2)

debug_library_cell

Debugs the specified library cell. This command is available only in the library verification mode.

SYNTAX

```
debug_library_cell
   [ cellName ]
```

Data Types

```
cellName string
```

ARGUMENTS

cellName

Specifies the name of cell that is to be debugged.

DESCRIPTION

Use this command to debug a library cell that has failed or aborted verification to obtain more information and analyze the problem. This command reverifies the specified library cell directly from the reference and implementation containers.

Use this command only after all cells are verified. It keeps the session in the *library_verify* mode.

SEE ALSO

```
verify(2)
select_cell_list(2)
```

report_cell_list(2)

define_design_lib

Defines the library mapping information.

SYNTAX

```
status define_design_lib
   [ -container containerID | -r | -i ]
   logical_library_name
   -path physical_library_name
```

Data Types

```
   containerID string
   logical_library_name string
   physical_library_name string
```

ARGUMENTS

-container *containerID*

Specifies the container from which the mapping is to be used. If you don't specify a container name, by default the current container is used.

-r

Specifies that the mapping in the default reference container is to be used.

-i

Specifies that the mapping in the default implementation container is to be used.

logical_library_name

Specifies the name of the logical library name.

-path *physical_library_name*

Specifies the name of the physical library including the path.

DESCRIPTION

You can use this command multiple times, once for each logical library name that is to be mapped to a physical library. You can run the commands at any time, but they are valid only before the first read into that logical library.

When multiple logical libraries point to the same physical library specified by using the *physical_library_name* argument, the tool uses the first logical library name that is defined. The path specified by using the **-path** option is used for proper mapping. All logical libraries that point to the same path, must share the same physical library.

A logical library cannot be mapped to two physical libraries in the same container. But they can be mapped to two physical libraries in different containers.

The **define_design_lib** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example maps logical libraries lib1 and lib2 to physical library lib1 in container r.

```
fm_shell> define_design_lib -r lib1 -path s
1
fm_shell> define_design_lib -r lib2 -path s
1
fm_shell>
```

The following example reports an error when a container is not available.

```
fm_shell> define_design_lib lib1 -path s
The current container is not set
0
fm_shell>
```

The following example reports an error when the same logical library is mapped to different physical libraries in the same container.

```
fm_shell> define_design_lib -r lib1 -path s
1
fm_shell> define_design_lib -r lib1 -path p
Warning: For current container r, the logical library LIB1 is already mapped to
the physical library LIB1.
Skipping current mapping information. (FM-518)
0
fm_shell>
```

SEE ALSO

```
read_vhdl(2)
current_container(2)
```

define_primitive_pg_pins

Defines the power and ground pins or power-down-function pin of User Defined Primitive(UDP) cells.

SYNTAX

```
status define_primitive_pg_pins
   [-power power_pin_names
    -ground ground_pin_names]
   [-power_down power_down_function_pin_name]
   primitive_list
```

Data Types

```
   power_pin_names list
   ground_pin_names list
   power_down_function_pin_name string
   primitive_list list
```

ARGUMENTS

-power *power_pin_names*

Specifies the power pin list of given list of User Defined Primitive(UDP) cells.

-ground *ground_pin_names*

Specifies the ground pin list of given list of User Defined Primitive(UDP) cells.

-power_down *power_down_function_pin_name*

Specifies the power-down-function pin of given list of User Defined Primitive(UDP) cells.

primitive_list

Specifies the list of power aware User Defined Primitive(UDP) cells. This can accept values which are in TCL glob-style pattern matching form.

DESCRIPTION

You can use this command to provide information on the power and ground pins or power-down-function pin of given list of User Defined Primitive(UDP) cells. This information assist the tool in identifying the power-down-function logic of these User Defined Primitive(UDP) cells.

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following examples show the usage of **define_primitive_pg_pins** command.

```
fm_shell> define_primitive_pg_pins -power VDD -ground VSS "udp_cell_1* udp_cell_2*"
1
fm_shell> define_primitive_pg_pins -power {VDD VDD1} -ground "VSS VSS1" "udp_cell*"
1
fm_shell> define_primitive_pg_pins -power_down PDF "udp_cell_3* udp_cell_4*"
1
fm_shell>
```

SEE ALSO

define_proc_attributes

Defines attributes of a Tcl procedure, including an information string for help, a command group, a set of argument descriptions for help, and so on. The command returns the empty string.

SYNTAX

```
string define_proc_attributes
   proc_name
   [-info info_text]
   [-define_args arg_defs]
   [-define_arg_groups group_defs]
   [-command_group group_name]
   [-return type_name]
   [-hide_body]
   [-hidden]
   [-dont_abbrev]
   [-permanent]
```

Data Types

proc_name	string
info_text	string
arg_defs	list
group_defs	list
group_name	string
type_name	string

ARGUMENTS

proc_name

Specifies the name of the existing procedure.

-info *info_text*

Provides a help string for the procedure. This is printed by the **help** command when you request help for the procedure. If you do not specify *info_text*, the default is "Procedure".

-define_args *arg_defs*

Defines each possible procedure argument for use with **help -verbose**. This is a list of lists where each list element defines one argument.

-define_arg_groups ***group_defs***

Defines argument checking groups. These groups are checked for you in parse_proc_arguments. each list element defines one group

-command_group ***group_name***

Defines the command group for the procedure. By default, procedures are placed in the "Procedures" command group.

-return ***type_name***

Specifies the type of value returned by this proc. Any value may be specified. Some applications use this information for automatically generated dialogs etc. Please see the type_name option attribute described below.

-hide_body

Hides the body of the procedure from **info body**.

-hidden

Hides the procedure from **help** and **info proc**.

-dont_abbrev

Specifies that the procedure can never be abbreviated. By default, procedures can be abbreviated, subject to the value of the **sh_command_abbrev_mode** variable.

-permanent

Defines the procedure as permanent. You cannot modify permanent procedures in any way, so use this option carefully.

-deprecated

Defines the procedure as deprecated i.e. it should no longer be called but is still available.

-obsolete

Defines the procedure as obsolete i.e. it used to exist but can no longer be called.

DESCRIPTION

The **define_proc_attributes** command associates attributes with a Tcl procedure. These attributes are used to define help for the procedure, locate it in a particular command group, and protect it.

When a procedure is created with the **proc** command, it is placed in the Procedures command group. There is no help text for its arguments. You can view the body of the procedure with **info body**, and you can modify the procedure and its attributes. The **define_proc_attributes** command allows you to change these aspects of a procedure.

Note that the arguments to Tcl procedures are all named, positional arguments. They can be programmed with default values, and there can be optional arguments by using the special argument name *args*. The

define_proc_attributes command does not relate the information that you enter for argument definitions with **-define_args** to the actual argument names. If you are describing anything other than positional arguments, it is expected that you are also using **parse_proc_arguments** to validate and extract your arguments.

The *info_text* is displayed when you use the **help** command on the procedure.

Use **-define_args** to define help text and constraints for individual arguments. This makes the help text for the procedure look like the help for an application command. The value for **-define_args** is a list of lists. Each element has the following format:

```
arg_name option_help value_help data_type attributes
```

The elements specify the following information:

- *arg_name* is the name of the argument.
- *option_help* is a short description of the argument.
- *value_help* is the argument name for positional arguments, or a one word description for dash options. It has no meaning for a Boolean option.
- *data_type* is optional and is used for option validation. The *data_type* can be any of: string, list, boolean, int, float, or one_of_string. The default is string.
- *attributes* is optional and is used for option validation. The *attributes* is a list that can have any of the following entries:
    - "required" - This argument must be specified. This attribute is mutually exclusive with optional.
    - "optional" - Specifying this argument is optional. This attribute is mutually exclusive with "required."
    - "value_help" - Indicates that the valid values for a one_of_string argument should be listed whenever argument help is shown.
    - "values {<list of allowable values>}" - If the argument type is one_of_string, you must specify the "values" attribute.
    - "type_name <name>" - Give a descriptive name to the type that this argument supports. Some applications may use this information to provide features for automatically generated dialogs, etc. Please see product documentation for details. This attribute is not supported on boolean options.
    - "merge_duplicates" - When this option appears more than once in a command, its values are concatenated into a list of values. The default behavior is that the right-most value for the option specified is used.
    - "remainder" - Specifies that any additional positional arguments should be returned in this option. This option is only valid for string option types, and by default the option is optional. You can require at least one item to be specified by also including the required option.
    - "deprecated" - Specifying this option is deprecated i.e. it should no longer be used but is still available. A warning will be output if this option is specified. This attribute cannot be combined with obsolete or required.
    - "obsolete" - Specifying this option is obsolete i.e. it used to exist but can no longer be used. A

warning will be output if this option is specified. This attribute cannot be combined with deprecated or required.

- "min_value" <value> - Specify the minimum value for this option. This attribute is only valid for integer and float types.
- "max_value" <value> - Specify the maximum value for this option. This attribute is only valid for integer and float types.
- "default" <value> - Specify the default value for this option. This attribute is only valid for string, integer and float option types. If the user does not specify this option when invoking the command this default value will be automatically passed to the associated tcl procedure.

The default for *attributes* is "required."

Use the **-define_arg_groups** to define argument checking groups. The format of this option is a list where each element in the list defines an option group. Each element has the following format:

{<type> {<opt1> <opt2> ...} [<attributes>]}

The types of groups are

- *"exclusive"* Only one option in an exclusive group is allowed. All the options in the group must have the same required/optional status. This group can contain any number of options.
- *"together"* If the first option in the group is specified then the second argument is also required. This type of group can contain at most two options.
- \fi"related" These options are related to each other. An automatic dialog builder for this command may try to group these options together.

The supported attributes are:

- *{"label" <text>}* An optional label text to identify this group. The label may be used by an application to automatically build a grouping in a generated dialog.
- *"bidirectional"* This is only valid for a together group. It means that both options must be specified together.

Change the command group of the procedure using the **-command_group** command. Protect the contents of the procedure from being viewed by using **-hide_body**. Prevent further modifications to the procedure by using **-permanent**. Prevent abbreviation of the procedure by using **-dont_abbrev**.

EXAMPLES

The following procedure adds two numbers together and returns the sum. For demonstration purposes, unused arguments are defined.

```
prompt> proc plus {a b} { return [expr $a + $b]}

prompt> define_proc_attributes plus -info "Add two numbers" \
? -define_args {
  {a "first addend" a string required}
```

```
      {b "second addend" b string required}
      {"-verbose" "issue a message" "" boolean optional}}

    prompt> help -verbose plus
    Usage: plus    # Add two numbers
        [-verbose]         (issue a message)
        a                  (first addend)
        b                  (second addend)

    prompt> plus 5 6
    11
```

In the following example, the argHandler procedure accepts an optional argument of each type supported by **define_proc_attributes**, then displays the options and values received. Note the only one of -Int, -Float, or -Bool may be specified to the command:

```
    proc argHandler {args} {
      parse_proc_arguments -args $args results
      foreach argname [array names results] {
        echo $argname = $results($argname)
      }
    }

    define_proc_attributes argHandler \
      -info "Arguments processor" \
      -define_args {
         {-Oos "oos help"       AnOos   one_of_string
            {required value_help {values {a b}}}}
         {-Int "int help"       AnInt   int     optional}
         {-Float "float help"   AFloat  float   optional}
         {-Bool "bool help"     ""      boolean optional}
         {-String "string help" AString string  optional}
         {-List "list help"     AList   list    optional}
         {-IDup "int dup help"  AIDup   int     {optional merge_duplicates}}
       } \
      -define_arg_groups {
         {exclusive {-Int -Float -Bool}}
       }
```

SEE ALSO

```
help(2)
info(2)
parse_proc_arguments(2)
proc(2)
sh_command_abbrev_mode(3)
```

diagnose

Runs diagnosis on the most recent verification that failed. By default, this command diagnoses the implementation design.

SYNTAX

```
integer diagnose
   [failing_compare_points]
   [-pattern_limit limit]
   [-effort_level low | medium | high]
   [-r]
   [-all]
```

Data Types

```
failing_compare_points string
limit integer
```

ENABLED SHELL MODES

Verify

ARGUMENTS

failing_compare_points

Specifies a list of failing compare points to be diagnosed in the following form:

```
{cp1 cp2 ...}
```

Use wildcard characters to match multiple points. By default, this command considers all compare points.

-pattern_limit *limit*

Specifies the maximum number of failing patterns to consider during diagnosis. The limit must be an integer. The default for the *limit* argument is 256 patterns. For more information about failing patterns, see the *Formality User Guide*.

`-effort_level low | medium | high`

Specifies the effort level for the diagnosis engine. The default effort level is **medium**.

`-r`

Diagnoses the reference design. By default, the command diagnoses the implementation design.

`-all`

Diagnoses all failing compare points.

DESCRIPTION

This command runs diagnosis on the most recent verification that failed.

If you run the **diagnose** command before performing verification, the tool reports the following error:

```
Error: You must perform verification before diagnosis
```

Use the list of error candidates that the command identifies to isolate problem areas in a failing design. A successful diagnosis identifies error candidates with single or multiple errors. Use the **report_error_candidates** command to view error candidates. For conceptual information about error candidates, see the *Formality User Guide*.

The **diagnose** command also identifies the corresponding matching regions in the matched design. For example, if you diagnosed an implementation design, matching regions are identified in the reference design. Set the *diagnosis_enable_find_matching_regions* Tcl variable to **false** to turn off matching regions.

Interrupting the **diagnose** command stops processing and any diagnosis results are lost. The tool does not retain partial diagnosis results.

During diagnosis, the tool uses a maximum of 256 failing patterns, by default, that apply to the failing compare points. Use the **-pattern_limit** option to use a maximum failing pattern limit other than the default or the value set by using the **diagnosis_pattern_limit** Tcl variable.

For best error resolution, diagnose the whole design without specifying any arguments. This might sometimes fail to return any potential errors, such as the FM-417 or FM-420 errors. In such cases, to diagnose a group of failing compare points, specify a list of compare points. If this also fails, diagnose a single compare point. For single compare points, the command diagnoses only the logic cone that defines the specified compare point.

EXAMPLES

The following example shows a failed verification and the subsequent diagnosis. The transcript notes the associated designs and provides a verification and diagnosis progress report. The number of failing patterns applied over the entire design is the maximum of 256.

```
prompt> verify
Reference design is 'r:/WORK/chkblk'
Implementation design is 'i:/WORK/chkblk'

********************************** Matching Results **********************************


 353 Compare points matched by name
 .br
 0 Compare points matched by signature analysis
 .br
 0 Compare points matched by topology
 .br
 173 Matched primary inputs, black box outputs
 .br
 0(0) Unmatched reference(implementation) compare points
 .br
 0(0) Unmatched reference(implementation) primary inputs, black box outputs
 .br
**************************************************************************************



Status:  Verifying...


    Compare point bd_gop_hdr_exist_reg failed (is not equivalent)


    Compare point bd_au_count_reg_0_ failed (is not equivalent)


    Compare point bd_au_count_reg_1_ failed (is not equivalent)


    Compare point bd_au_count_reg_2_ failed (is not equivalent)


    Compare point bd_au_count_reg_3_ failed (is not equivalent)


    Compare point bd_au_count_reg_4_ failed (is not equivalent)


    Compare point bd_au_count_reg_5_ failed (is not equivalent)
```

********************************** Verification Results **********************************
Verification FAILED
ATTENTION: RTL interpretation messages were produced during link
of reference design.
Verification results may disagree with a logic simulator.

Reference design: r:/WORK/chkblk
Implementation design: i:/WORK/chkblk

346 Passing compare points
7 Failing compare points
0 Aborted compare points

Matched Compare Points BBPin Loop BBNet Cut Port DFF LAT TOTAL

Passing (equivalent) 0 0 0 0 137 209 0 346
Failing (not equivalent) 0 0 0 0 0 7 0 7
Not Compared
Constant reg 3 0 3
****

```
prompt> diagnose
Status:  Diagnosing i:/WORK/chkblk vs r:/WORK/chkblk...
Status:  Diagnosis initializing...
Status:  Analyzing patterns...
Single error detected in implementation.
    Single error detected in implementation design.
    Number of error candidates: 3
    Analysis completed
Status:  Finding matching regions in reference design...
    Single matching region detected in reference design.
Diagnosis completed
1
fm_shell (verify)>
```

SEE ALSO

```
report_diagnosed_matching_regions(2)
report_error_candidates(2)
report_failing_points(2)
report_passing_points(2)
```

discard_edits

Removes all design copies created by **edit_design**.

SYNTAX

```
discard_edits
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command removes all the edited designs created by the **edit_design** command. Any edits done are lost unless they have previously been applied using the **apply_edits** or **verify_edits** commands.

EXAMPLES

Edit a design and then discard the edits:

```
fm_shell (verify)> edit_design i:/WORK/bot
fm_shell (verify)> create_net ECO_NET_1
fm_shell (verify)> discard_edits
```

SEE ALSO

```
apply_edits(2)
create_net(2)
edit_designs(2)
verify_edits(2)
```

disconnect_net

Disconnects a net from one or more pins.

SYNTAX

```
status disconnect_net
   net_name
   pin_list
   -all
```

Data Types

```
      net_name string
      pin_list list
```

RETURN VALUE

The **disconnect_net** command returns a status of 1 if it was successful and 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

net_name

Diconnects the specific net. You can specify either an object ID or an instance-based path.

The command disconnects nets in a design and the changes are visible to all instances of the design. To only affect a specific instance-based path, uniquify the instance of the design before running this command.

pin_list

Disconnects the specified pins from the net. You can specify either object IDs or instance-based paths. You must specify either the *pin_list* or the *-all* option.

-all

Disconnects all pins from a net. You must specify either the *pin_list* or the *-all* option.

DESCRIPTION

This command removes the connections between a net and one or more pins. The net and pins are not removed.

You can specify pins that are directly on the given net segment itself or pins that are connected to other segments of the net.

The *-all* option removes the pins directly connected to the given net without considering any other segments of the net.

To connect pins to a net, use the **connect_net** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example shows how to disconnect all pins from net i1 and two pins from net i2.

```
fm_shell (setup)> current_instance r:/WORK/top/m1
fm_shell (setup)> disconnect_net i1 -all
fm_shell (setup)> disconnect_net i2 {b1/in1 b2/out1}
```

SEE ALSO

```
connect_net(2)
current_design(2)
current_instance(2)
edit_design(2)
```

echo

Displays the values of the specified arguments.

SYNTAX

```
echo
    [-n] [argument]
```

Data Types

```
argument string
```

ARGUMENTS

-n

Suppresses the default new-line behavior.

argument

Specifies the arguments to be displayed.

DESCRIPTION

This command returns the values of the given arguments. Use the **echo** command to display the value of variables, expressions, and text strings. When you specify a list of arguments, separate them with a space.

By default, each value appears on a new line. When you use the **-n** switch, multiple values are printed in the same line. To redirect the output, use the ">" and ">>" operators.

EXAMPLES

```
fm_shell> echo "Running version" $sh_product_version
Running version v3.0a
fm_shell> echo -n "Printing to" [expr (3 - 2)] > foo
fm_shell> echo " line." >> foo
fm_shell> sh cat foo
Printing to 1 line.
```

SEE ALSO

sh(2)

edit_design

Edit a copy of the specified design.

SYNTAX

```
edit_design
   design
   [ -hierarchy ]
```

Data Types

```
design string
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

design

Specifies the name of the design to edit.

-hierarchy

Make editable copies of all the designs instantiated by "*design*" recursively down the hierarchy.

DESCRIPTION

This command creates a copy of the specified design. The copy will be placed in a separate library meant for modifications via edit commands (such as **create_net** and **disconnect_net**). The design is only copied if it has not already been copied for edit.

This command sets the **current_design** to the editable design.

When it returns to setup mode, Formality will either apply the edits to the original designs, or discard the edited designs. See the man page for the **setup** command for details.

EXAMPLES

Edit a design:

```
fm_shell (verify)> edit_design i:/WORK/bot
fm_shell (verify)> create_net ECO_NET_1
```

SEE ALSO

```
create_net(2)
disconnect_net(2)
setup(2)
```

elaborate_library_cells

Resolves cell references before verifying the library. This command is available only in the library verification mode.

SYNTAX

```
elaborate_library_cells
```

DESCRIPTION

Use this command to link cell instances after reading in both libraries. Apply verification constraints after this command has completed. If constraints are not necessary, use the **verify** command to complete the verification of the libraries after they both are read in.

EXAMPLES

This example elaborates cells in both libraries.

```
fm_shell> elaborate_library_cells
```

SEE ALSO

```
verify(2)
```

elapsed_time

Returns the number of seconds of wall clock time elapsed since Formality started.

SYNTAX

```
elapsed_time
```

ARGUMENTS

none

DESCRIPTION

This command returns the amount of wall clock time, in seconds, elapsed since Formality started.

EXAMPLES

The following example shows the output produced by the **elapsed_time** command.

```
fm_shell (match)> elapsed_time
1342
fm_shell (match)>
```

end_alternate_strategies

End alternate strategies runs.

SYNTAX

```
end_alternate_strategies
   -directory pathname
   [ -strategies strategies_list ]
```

Data Types

```
pathname string
strategies_list string
```

ARGUMENTS

-directory *pathname*

Specifies the directory where the strategies are being run.

-strategies *strategies_list*

Specifies the list of strategies to be stopped.

DESCRIPTION

This command provides a way of stopping some or all of the alternate strategies that were run using the **run_alternate_strategies** command.

The option -directory is required to identify the run that is being stopped.

If option -strategies is not specified, all the strategies are stopped.

EXAMPLES

The following example ends all alternate strategies runs in directory ras.

```
fm_shell> end_alternate_strategies -directory ras
```

SEE ALSO

```
run_alternate_strategies(2)
```

error_info

Displays detailed information on errors that are reported by the previous command.

SYNTAX

```
error_info
```

DESCRIPTION

This command displays detailed information about errors reported by the previous command. When an error occurs, the **error_info** command helps you identify the exact line in the block that caused the error.

EXAMPLES

In this example, the iterator variable "s" is not dereferenced in the 'if' statement. It should be "$s == "a" ".

```
fm_shell> foreach s $my_list {
?         if { s == "a" } {
?           echo "Found 'a'!"
?         }
?       }
Error: syntax error in expression " s == "a" "
       Use error_info for more info. (CMD-013)
fm_shell> error_info
Extended error info:
syntax error in expression " s == "a" "
    while executing
"if { s == a } {
     echo "Found 'a'"
}"
    ("foreach" body line 2)
    invoked from within
"foreach s [list a b c] {
   if { s == a } {
     echo "Found 'a'"
}
}"
 -- End Extended Error Info
```

exit

Terminates the application.

SYNTAX

```
integer exit
   [exit_code]
```

Data Types

```
exit_code      integer
```

ARGUMENTS

exit_code

Specifies the return code to the operating system. The default value is 0.

DESCRIPTION

This command exits from the application. You have the option to specify a code to return to the operating system.

EXAMPLES

The following example exits the current session and returns the code 5 to the operating system. At a UNIX operating system prompt, verify (**echo**) the return code as shown.

```
prompt> exit 5
```

```
    % echo $status
    5
```

SEE ALSO

```
quit(2)
```

filter_collection

Filters an existing collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection filter_collection
   [-regexp]
   [-nocase]
   collection1
   expression
```

Data Types

collection1	collection
expression	string

ARGUMENTS

-regexp

Specifies that the =~ and !~ filter operators will use real regular expressions. By default, the =~ and !~ filter operators use simple wildcard pattern matching with the * and ? wildcards.

-nocase

Makes the pattern match case-insensitive.

collection1

Specifies the base collection to be filtered. This collection is copied to the result collection. Objects are removed from the result collection if they are evaluated as **false** by the conditional *expression* value. Substitute the collection you want for *collection1*.

expression

Specifies an expression with which to filter *collection1*. Substitute the string you want for *expression*.

DESCRIPTION

Filters an existing collection, resulting in a new collection. The base collection remains unchanged.

In many cases, application commands that create collections support a *-filter* option that filters as part of the collection process, rather than after the collection has been made. This type of filtering is almost always more efficient than using the **filter_collection** command after a collection has been formed. The **filter_collection** command is most useful if you plan to filter the same large collection many times using different criteria.

The **filter_collection** command results in either a new collection or an empty string. A resulting new collection contains the subset of the objects in the input *collection1*. A resulting empty string (the empty collection) indicates that the *expression* filtered out all elements of the input *collection1*.

The basic form of the conditional expression is a series of relations joined together with AND and OR operators. Parentheses () are also supported. The basic relation contrasts an attribute name with a value through a relational operator. For example,

```
is_inverted == true and direction == in
```

The value side of a relation can be a simple string, quoted string, or an attribute name prefixed with "@". For example:

```
input_delay<=@output_delay
input_delay<=0.24
name=="@literal_name"
```

The relational operators are

```
==   Equal
!=   Not equal
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
=~   Matches pattern
!~   Does not match pattern
```

The basic relational rules are

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only either *true* or *false*.

The **filter_collection** command has a *-regexp* option that uses regular expressions when matching for string attributes. Regular expression matching is done in the same way as in the Tcl **regexp** command. When using the *-regexp* option, take care in the way you quote the filter *expression*. Using rigid quoting with curly braces around regular expressions is recommended.

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 simply by adding ".*" to the beginning or end of the expressions as needed. You can make the regular expression search case-insensitive by using the *-nocase* option.

EXAMPLES

The following example from Formality creates a collection of only techlib cells.

```
prompt> set a [filter_collection \
?         [get_cells *] "is_techlib == true"]
{r:/WORK/top/U1 r:/WORK/top/U2}
```

The following example from Formality does the same as above, but using the *-filter* option of the **get_cells** command.

```
prompt> set a [get_cells * -filter "is_techlib == true"]
{r:/WORK/top/U1 r:/WORK/top/U2}
```

The following shows how to use the *-filter* option of the **get_cells** command to only return the AND gates.

```
prompt> get_cells $ref/* -filter {@cell_type == "AND"}
{r:/WORK/top/C1 r:/WORK/top/C2}
```

SEE ALSO

```
collections(2)
get_cells(2)
regexp(2)
```

find_cells

Lists cells in the current design.

SYNTAX

```
collection find_cells
   [-of_objectID objectID]
   [-library_cells | -nolibrary_cells]
   [-type ID_type ]
   [cellID_list]
```

Data Types

```
objectID string
ID_type string
cellID_list string
```

ARGUMENTS

-of_objectID objectID

Specifies a pin to which returned cells are connected or specifies the design in which the returned cells are located.

-library_cells

Include cells referenced from technology library cells in the cell list.

-nolibrary_cells

Includes the cells that are not referenced from technology library cells in the cell list.

-type *ID_type*

Resolves conflicts between design objects specified by using the **-of_object** option that are of different types but have the same name. Specify one of the following ID types:

- *pin* for a pin type
- *design* for a design

cellID_list

Lists object IDs in the current design. Each object ID must resolve to an object of type *cell*.

DESCRIPTION

This command lists cells in the current design.

If you use this command without specifying a list of cell names or object IDs, the command lists all the cells in the current design by default.

The tool returns the following string if you have not established a current design:

```
Error:  The current design is not set. A design must be specified.
```

To return a list of cells, specify a list of object IDs that are defined as cell types. To list cells that are connected to specific object IDs, use the **-of_objectID** option. For example, to list cells attached to a specific pin:

```
find_cells  -of_objectID objectID
```

To limit the list of cells that are reported by this command to those referenced or not referenced from a technology library, use the **-library_cells** switch or the **-nolibrary_cells** switch, respectively.

If you use the **find_cells** command with the **-of_objectID** option and specify a design ID argument, the tool returns a list of cells in the current design that are instantiated from the specified design ID.

If you use the **-of_objectID** option and you know that naming conflicts exist for the specified *objectID*, you can use the **-type** option to resolve those objects.

The **find_cells** command returns a list of cell names.

EXAMPLES

The following examples demonstrate how to use the **find_cells** command to obtain information about cells in the current design. In these examples the current design is named */WORK/mR4000*.

The following example returns a list of all cells.

```
prompt> find_cells {ref:/WORK/mR4000/Instruction_reg[0]}
ref:/WORK/mR4000/register
```

The following example lists cells that are instantiations of a design named **mA1u**.

```
prompt> find_cells -of_objectID ref:/*/mA1u
ref:/WORK/mR4000/alu
```

The following example lists cells in a design named **mA1u**.

```
prompt> find_cells ref:/*/mA1u/*/
ref:/WORK/mA1u/U403 ... ref:/WORK/mA1u/r23
```

The following example lists cells that are not references of technology library cells.

```
prompt> find_cells -nolibrary_cells
ref:/WORK/mR4000/alu ... ref:/WORK/mR4000/register
```

SEE ALSO

```
current_design(2)
find_nets(2)
find_pins(2)
find_ports(2)
find_references(2)
list_libraries(2)
```

find_compare_points

Returns a list of compare points affected by the specified design objects.

SYNTAX

```
find_compare_points
     -edits | objectID_list | -edits objectID_list
   [ -list | -collection ]
   [ -type ID_type ]
   [ -status status_type ]
```

Data Types

```
ID_type string
status_type string
objectID string
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-edits

Identifies fan out compare points from edits that are made using editing commands such as **create_cell** and **connect_net**. When combined with an *objectID_list* the compare point fan out is determined from edits in designs specified by the *objectID_list* argument. You cannot use the *-type* option with the **-edits** option.

objectID_list

Specifies design objects from which to start looking for compare points when the \fi-edits option is not specified. If you specify a name consisting of a wildcard expression that resolves to more than one object, the operation is applied to all the matching objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is used in the following precedence: pin, port, net, cell.

-list

Reports compare points as a Tcl list. You can use this list in further Tcl processing.

-collection

Reports compare points as a collection. You can use this collection in further Tcl processing.

-type *ID_type*

Specifies the type of the objects to use from the *objectID_list*. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following object types:

- *pin* - to specify pin type
- *port* - to specify port type
- *net* - to specify net type
- *cell* - to specify cell type

-status *status_type*

Limits the compare points to those with a specific status. Specify one of the following:

- *passing* - to specify passing compare points
- *failing* - to specify failing compare points
- *aborted* - to specify aborted compare points
- *unverified* - to specify unverified compare points
- *unmatched* - to specify unmatched compare points
- *not_compared* - to specify disabled compare points

DESCRIPTION

This command returns a report of the compare points which are in the fan out of the specified ObjectIDs or edits. By default, the command returns a report with one compare point per line. This command is useful for ECO modifications, where a change to one or more design objects is made and you want to find the affected compare points. You can then verify the affected points using multiple point verification with the **set_verify_points** command.

If the **-list** option is specified, the command returns a Tcl list of compare points.

By default, the **find_compare_points** command returns compare points regardless of their status. However, you can limit the search to only compare points with certain statuses using one or more **-status** options.

EXAMPLES

The following example returns all the compare points that structurally depend on a given design object.

```
fm_shell> find_compare_points r:/WORK/top/n123
 (DFF)     r:/WORK/top/mult1/mult2/prod_preiso_reg[7]
 (DFF)     r:/WORK/top/mult1/mult2/prod_preiso_reg[8]
 (DFF)     r:/WORK/top/mult1/mult2/prod_preiso_reg[9]
 (Port)    r:/WORK/top/prod[0]
 (Port)    r:/WORK/top/prod[10]
```

This example shows how to obtain a Tcl list of all compare points affected by edits.

```
fm_shell> find_compare_points -edits -list
{i:/WORK/top/sum[11]}  {i:/WORK/top/sum[12]}  {i:/WORK/top/sum[13]}
{i:/WORK/top/sum[14]} {i:/WORK/top/sum[15]}
```

This example shows how to report all compare points affected by edits in the design i:/WORK/block142.

```
fm_shell> find_compare_points -edits i:/WORK/block142
 (DFF)     i:/WORK/block142/prod[0]
 (DFF)     i:/WORK/block142/prod[1]
 (Port)    i:/WORK/block142/early[0]
 (Port)    i:/WORK/block142/early[1]
```

This example shows how to obtain a Tcl list of all failing and unverified compare points affected by edits.

```
fm_shell> find_compare_points -edits -list -status failing -status unverified
{i:/WORK/top/sum[13]} {i:/WORK/top/sum[14]} {i:/WORK/top/sum[15]} {i:/WORK/top/carry1}
```

SEE ALSO

```
set_verify_points(2)
verify_edits(2)
```

find_designs

Returns a list of designs in the current design.

SYNTAX

```
collection find_designs
   [object_list]
   [-passing]
   [-failing]
   [-aborted]
   [-not_verified]
```

Data Types

```
object_list string
```

ARGUMENTS

object_list

Specifies objects in the current workspace.

-passing

Specifies designs that have passed the most recent verification.

-failing

Specifies designs that have failed the most recent verification.

-aborted

Specifies designs that have aborted during the most recent verification.

-not_verified

Specifies designs that are not verified during the most recent verification.

DESCRIPTION

This command returns a list of designs when you specify an object list. If you do not specify a list of designs or object IDs, the tool lists all the designs that are loaded.

To list designs with a specific verification status, specify the appropriate switch. The verification status is based on the most recent verification. If verification is not performed, the verification status of all designs is "not verified."

The **-passing**, **-failing**, and **-aborted** switches are applicable only to implementation designs because the tool verifies implementation designs. The **-not_verified** switch is applicable to all designs.

If a matching design is not found, the command does not return anything. Otherwise, the command returns one of the following:

- 0 to indicate failure
- List of designs names for success

EXAMPLES

The following example lists all designs currently loaded. Due to the large number of designs, this example shows only a partial listing.

```
prompt> find_designs -failing
impl:/WORK/impl impl:/WORK/myor2 impl:/WORK/myor2_reg

prompt> find_designs -passing
impl:/WORK/myopt2 impl:/WORK/myreg

prompt> find_designs -not_verified
impl:/WORK/myand2 impl:/WORK/myand2_reg ...

prompt> find_designs -passing impl:/*/*opt*
impl:/WORK/myopt2

...

prompt>
```

SEE ALSO

```
find_cells(2)
find_pins(2)
find_ports(2)
find_references(2)
list_libraries(2)
report_designs(2)
```

find_drivers

Returns a list of drivers of the specified net.

SYNTAX

```
status find_drivers
   [-hier]
   net_name
```

Data Types

```
net_name string
```

RETURN VALUE

The **find_drivers** command returns a list of drivers, each specified by absolute path, of the given net, or 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-hier

Lists drivers across the specified hierarchy.

net_name

Lists the drivers of the specified net. The net must be specified as either an absolute or relative to (**current_instance**) the instance-based path.

DESCRIPTION

The command finds and returns a list of all drivers of the net. All technology library cell output pins or top-level input ports of the net segment (or across hierarchy, if "-hier" is specified) are returned. Note that if the **-hier** option is not specified, cellpins are returned.

EXAMPLES

The following example shows how to find all local drivers of net r:/WORK/top/M1/B1/o1.

```
fm_shell > find_drivers r:/WORK/top/M1/n1
r:/WORK/top/M1/B1/o1

The following example shows how to drivers across hierarchy for the same net
fm_shell > find_drivers -hier r:/WORK/top/M1/n1
r:/WORK/top/M1/B1/C1/Z

Note that the same results could be obtained using current_instance and relative paths:
fm_shell > current_instance r:/WORK/top
fm_shell > find_drivers M1/n1
r:/WORK/top/M1/B1/o1
```

SEE ALSO

```
find_receivers(2)
current_instance(2)
create_net(2)
connect_net(2)
```

find_equivalent_nets

Finds nets in the implementation design that are equivalent to the specified nets.

SYNTAX

```
status find_equivalent_nets
   [ -nets candidateNet_list ]
   [ -list ]
   [ -fanin ]
   [ inputNet_list ]
```

Data Types

```
candidateNet_list list
inputNet_list list
```

Enabled Shell Modes

Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-nets candidateNet_list

Restricts the search for equivalent nets to the specified list of nets. The candidate nets must be from the implementation design. Any other net in this list is ignored. If this option is not specified, the command finds all equivalent nets in the implementation design.

-list

Reports equivalent nets as a Tcl list. You can use this list in further Tcl processing.

-fanin

Look up-stream for nets with equivalences, if no equivalences are found.

inputNet_list

Specifies a list of nets whose equivalences are to be found. This list of nets can be any combination of reference and implementation nets.

DESCRIPTION

This command finds nets in the implementation design that are equivalent to the specified nets. The input net can be either a reference or implementation net. An equivalent net can either be positively (identical) or negatively (inverse) equivalent to the input net.

By default, the command searches the entire implementation design for all instances of nets that are equivalent to the input net. You can restrict the search to specific nets using the **-nets** option. The list of nets must be implementation design nets. Any other net in the candidate list is ignored. The **find_region_of_nets** command can be used to build lists of candidate nets.

The default output of the command is a report that lists each input net followed by its equivalences. The input net together with its equivalences form an equivalence class. The **-list** option produces a Tcl list output of equivalence classes which is amenable for further processing in a Tcl script.

The format of an equivalence class is as follows:

```
<input_net> <class_type> { <equiv_net1> <polarity> <equiv_net2> <polarity> ... }
```

<*input_net*> is the input net whose equivalences were found

The <*class_type*> field is one of the following values:

+

Normal equivalence class

?

The equivalence class timed out or was interrupted by the user. All equivalences may not be listed. A timeout value for **find_equivalent_nets** is set using the **equivalent_nets_timeout_limit** variable.

0

The input net of this class is a constant 0

1

The input net of this class is a constant 1

X

The input net of this class is a constant X

The list following <*class_type*> consists of pairs of equivalent nets and its polarity with respect to the input net.

The <*polarity*> field can be one of the following:

+

The equivalent net function is identical to the input net function.

-

The equivalent net function is the inverse of the input net function.

?

The equivalent net and the input net are highly similar, but Formality could not conclusively prove equivalence.

EXAMPLES

The following examples show how to use the **find_equivalent_nets** command.

```
fm_shell (match)> find_equivalent_nets -list r:/WORK/m1/b1/r1
{ r:/WORK/m1/b1/r1 + { r:/WORK/m1/b1/r2 + i:/WORK/m1/b1/r1 - } }

fm_shell (match)> find_equivalent_nets -list r:/WORK/m1/b1/r1
{ r:/WORK/m1/b1/r1 ? { r:/WORK/m1/b1/r2 + r:/WORK/m1/b2/r3 - i:/WORK/m1/b1/r1 ? } }
```

The following example shows a report of nets that are equivalent to *r:/WORK/rtl1/PORTB[1]*:

```
fm_shell (verify)> find_equivalent_nets r:/WORK/rtl1/PORTB[1]
--- Equivalent Nets:
    Ref  Net  + r:/WORK/rtl1/PORTB[1]
   Impl  Net  + i:/WORK/rtl1/U27/B
   Impl  Net  + i:/WORK/rtl1/U11/A
   Impl  Net  + i:/WORK/rtl1/U7/A
   Impl  Net  + i:/WORK/rtl1/U36/B
   Impl  Net  + i:/WORK/rtl1/PORTB[1]
```

The following example shows how to use the *-nets* option to constrain the search space of candidate equivalent nets to implementation nets in failing cones:

```
fm_shell (verify)> set region [find_region_of_nets -implementation -failing]
fm_shell (verify)> find_equivalent_nets -nets $region r:/WORK/rtl1/PORTB[1]
--- Equivalent Nets:
    Ref  Net  + r:/WORK/rtl1/PORTB[1]
   Impl  Net  + i:/WORK/rtl1/U27/B
   Impl  Net  + i:/WORK/rtl1/U11/A
```

SEE ALSO

```
equivalent_nets_timeout_limit(3)
find_region_of_nets(2)
```

find_nets

Returns a list of nets in the current design.

SYNTAX

```
find_nets
   object_list
   [ -hierarchy ]
   -of_objectID objectID
   -type ID_type
```

Data Types

```
   object_list string
   objectID string
   ID_type string
```

ARGUMENTS

object_list

Specifies one or more object IDs in the current design. For information on how to specify an object ID, see the *Formality User Guide*.

-hierarchy

Includes nets instantiated below the specified designs.

-of_objectID *object_ID*

Includes the nets connected to the specified pin or port. For information on how to specify an object ID, see the *Formality User Guide*.

-type *ID_type*

Resolves conflicts between design objects referenced with the *-of_object* option that are of different types but share the same name. Specify one of the following values for the *ID_type* argument:

- *pin* for a pin type
- *port* for a port type

DESCRIPTION

This command returns a list of nets.

To list nets connected to specific object IDs, specify the *-of_object* option. For example, to list nets attached to a specific pin, use the following command:

```
find_nets  -of_objectID objectID
```

When you use the *-of_objectID objectID* option, to resolve naming conflicts in the design, use the *-type* option.

When you don't specify a list of nets or object IDs, the command lists all nets by default.

The tool reports the following error if a current design is not established:

```
Error: The current design is not set. A design must be specified.
```

Otherwise, the **find_nets** command returns one of the following:

- 0 to indicate failure
- List of net names

EXAMPLES

The following example lists the nets in the current design. Due to the large number of nets, this example shows a partial listing.

```
fm_shell> find_nets
ref:/WORK/CORE/CC
ref:/WORK/CORE/CCEN
ref:/WORK/CORE/CIN
...
ref:/WORK/CORE/Y_9
ref:/WORK/CORE/ZERO
fm_shell>
```

SEE ALSO

```
find_cells(2)
find_pins(2)
find_ports(2)
find_references(2)
```

```
list_libraries(2)
```

find_pins

Returns a list of pins in the current design.

SYNTAX

```
find_pins
   [ -of_objectID objectID ]
   [ -in ]
   [ -out ]
   [ -inout ]
   [ -hierarchy ]
   [ -type type ]
   [pinID_list]
```

Data Types

```
objectID string
object_list string
type string
```

ARGUMENTS

-of_objectID *objectID*

Specifies a cell or net connected to the pins. For information on how to specify an object ID, see the *Formality User Guide*.

-in

Includes the input pins in the pin list.

-out

Includes the output pins in the pin list

-inout

Includes the bidirectional pins in the pin list.

-hierarchy

Includes pins that are connected at all levels of the hierarchy in the pin list.

-type *type*

Resolves conflicts between design objects referenced in the *-of_object* option that are of different types but have the same name. Specify one of the following values for the *ID_type* argument:

- *cell* for a cell type
- *net* for a net type

pinID_list

Specifies a list of pin names.

DESCRIPTION

This command returns a list of pins.

To find pins connected to a specific net or cell, use the *-of_object* option:

```
find_pins  -of_objectID objectID
```

If you use the *-of_objectID* option, and you know that naming conflicts exist for the specified *objectID*, use the *-type* option to resolve the conflicts.

Filter the list for pins of a certain type by specifying the *-in*, *-out*, or *-inout* options.

Use the *-hierarchy* switch to return a list of pins that crosses hierarchical bounds. This allows you to see every pin connected in a hierarchical design.

The tool lists all the pins when you use the **find_pins** command without specifying a list of pins or without using the *-of_objectID* option.

The tool reports the following error if a current design is not established:

```
Error: The current design is not set. A design must be specified.
```

Otherwise, the **find_pins** command returns one of the following:

- 0 to indicate failure
- List of pin names

EXAMPLES

The following example lists all pins in the current design. Due to the large number of pins, this example shows a partial listing.

```
fm_shell> find_pins
ref:/WORK/CORE/CC
```

```
    ref:/WORK/CORE/CCEN
    ref:/WORK/CORE/CIN
    ...
    ref:/WORK/CORE/Y_7
    ref:/WORK/CORE/Y_8
    ref:/WORK/CORE/Y_9
    fm_shell>
```

SEE ALSO

```
find_cells(2)
find_nets(2)
find_ports(2)
find_references(2)
list_libraries(2)
```

find_ports

Returns a list of ports in the current design.

SYNTAX

```
find_ports
   [ -of_objectID ID ]
   [ -in ]
   [ -out ]
   [ -inout ]
   [ -type type ]
   [portID_list]
```

Data Types

```
objectID string
object_list string
ID_type string
```

ARGUMENTS

-of_objectID *ID*

Specifies a net or design connected to the returned ports. For information on how to specify an object ID, see the *Formality User Guide*.

-in

Includes input ports in the list of ports.

-out

Includes output ports in the list of ports.

-inout

Includes bidirectional ports in the list of ports

-type *type*

Resolves conflicts between design objects referenced with the *-of_objectID* option that are of different types but have the same name. Specify one of the following values for the type ID:

- *net* for a net type
- *design* for a design

`portID_list`

Specifies a list of port names.

DESCRIPTION

This command returns a list of ports in the current design.

To find ports that are connected to a specific net or design, specify the *-of_objectID* option:

```
find_ports  -of_objectID objectID
```

If you use the *-of_objectID* option, and you know that naming conflicts exist for the specified *objectID*, use the *-type* option to help you resolve the conflicts.

Filter the list for pins of a certain type by specifying the *-in*, *-out*, or *-inout* switches.

When you use the **find_ports** command without specifying a list of ports or without using the -*of_objectID* option, it reports all the ports by default.

The tool reports the following error if the current design is not established:

```
Error:  The current design is not set. A design must be specified.
```

Otherwise,the **find_ports** command returns one of the following:

- 0 to indicate failure
- List of port names for success

EXAMPLES

The following example lists all ports in the current design. Due to the large number of ports, this example shows a partial listing.

```
fm_shell> find_ports
ref:/WORK/CORE/CC
ref:/WORK/CORE/CCEN
ref:/WORK/CORE/CIN
...
ref:/WORK/CORE/Y_7
ref:/WORK/CORE/Y_8
ref:/WORK/CORE/Y_9
fm_shell>
```

SEE ALSO

```
find_cells(2)
find_nets(2)
find_pins(2)
find_references(2)
list_libraries(2)
```

find_receivers

Returns a list of all receivers for the given net.

SYNTAX

```
status find_receivers
   [-hier]
   net_name
```

Data Types

```
net_name string
```

RETURN VALUE

The **find_receivers** command returns a list of receivers, each specified by absolute path, of the given net, or 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-hier

Lists receivers across the specified hierarchy.

net_name

Lists receivers of the specified net. This net must be specified as either an absolute or relative to instance-based path.

DESCRIPTION

This command lists receivers of the specified net. Technology library cell input pins or top-level output ports of the net segment (or across the specified hierarchy) are returned. Note that if the **-hier** option is not specified, the command lists cell pins.

EXAMPLES

The following example shows how to find all local receivers of net r:/WORK/top/M1/B1/o1.

```
fm_shell > find_receivers r:/WORK/top/M1/B1/o1
r:/WORK/top/M1/B1/o1

The following example shows how to receivers across hierarchy for the same net
fm_shell > find_receivers -hier r:/WORK/top/M1/B1/o1
r:/WORK/top/M1/C1/A

Note that the same results could be obtained using the current_instance command and relative
paths:
fm_shell > current_instance r:/WORK/top
fm_shell > find_receivers M1/B1/o1
r:/WORK/top/M1/B1/o1
```

SEE ALSO

```
find_drivers(2)
current_instance(2)
create_net(2)
connect_net(2)
```

find_references

Returns a list of designs instantiated within the current design.

SYNTAX

```
find_references
   [ -of_objectID cellID ]
   [ -hierarchy ]
   [ -black_box ]
   [designID_list]
```

Data Types

```
objectID string
design_list string
```

ARGUMENTS

designID_list

Specifies one or more design IDs from which to generate a list of instantiated designs. For information on how to specify a design ID, see the *Formality User Guide*.

-of_objectID *cellID*

Specifies the cell whose design name is returned. Each *objectID* must resolve to an object defined as a cell type.

-hierarchy

Includes all designs that are instantiated below the specified designs.

-black_box

Includes black boxes and unlinked devices.

designID_list

Specifies a list of design within which the tool will look for design references.

DESCRIPTION

This command returns a list of designs instantiated within the specified designs or within the current design.

When you use the **find_references** command without specifiying a list of *designIDs* or a particular cell's *objectID*, it reports all the designs within the current design, by default.

The command reports the following error if you have not established a current design:

```
Error:  The current design is not set. A design must be specified.
```

The **find_references** command returns one of the following:

- 0 to indicate failure
- List of designs for success

EXAMPLES

The following examples first set the current design and then generate a list of designs located within the current design. In these examples, the current design is /WORK/mR4000.

```
fm_shell> current_design ref:/*/mR4000
ref:/WORK/mR4000
fm_shell> find_references
ao4a2 ao7a1 ... oa1f0 oa1f1
```

The following example returns the designs instantiated within design mR4000.

```
fm_shell> find_references ref:/*/mA1u
and3b1 and8a1 ... or3b1 or4b1
```

The following example returns the designs referenced by cell mR4000/U160.

```
fm_shell> find_references -of_objectID ref:/*/mR4000/U160
buf1a4
```

The following example returns all design references at all levels of hierarchy. In this case, the instantiated designs within the current design are listed because the current design was not linked.

```
fm_shell> find_references -hierarchy
ao4a2 ao7a1 ... oa1f0 oa1f1
```

The following example lists design references that are unlinked or are black boxes.

```
fm_shell> find_references -hierarchy
ao4a2 ao7a1 ... oa1f0 oa1f1
```

In the preceding example, the instantiated designs within the current design are listed because the current design was not linked. If the current design was linked, the list would appear like this:

```
      ref:/CBA_CORE/and2b1 ... ref:/WORK/mRegister
```

SEE ALSO

```
find_cells(2)
find_nets(2)
find_pins(2)
find_ports(2)
list_libraries(2)
```

find_region_of_nets

Lists nets that define a region in the design.

SYNTAX

```
status find_region_of_nets
   [ -cells <cells_list> | -recursive ]
   [ -reference ]
   [ -implementation ]
   [ -failing ]
   [ comparePoint_list ]
```

Data Types

```
cells_list list
comparePoint_list list
```

Enabled Shell Modes

Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

comparePoint_list

Returns a collection of nets in the cones of the specified compare points. Matching compare points are implicitly included in this list.

-cells <cells_list>

Returns a collection of nets contained in the specified cells.

-recursive

Includes nets in cells contained within the specified cells when this option is specified with the *-cells* option.

-failing

Includes nets in the fan-in cones of the failing compare points.

-reference

Includes only reference nets.

-implementation

Includes only implementation nets.

DESCRIPTION

This command generates a collection of nets. The collection can be stored in Tcl variables and passed to other Formality commands such as **find_equivalent_nets**, for example to limit the search space to a particular region of the design.

Collections of nets can be computed in the following ways:

Using a List of Compare Points

When you specify a list of compare points using the **find_region_of_nets** command, it returns a collection of nets in the fan-in cone of each compare-point. The command also implictly includes nets from the matching compare-point cones.

From Cells

When you specify a list of cell names using the **-cells** option, the command returns a list of nets within the specified cells. Nets from sub-cells can be included by recursively descending down the hierarchy using the **-recursive** option.

From Failing Compare Points

You can list nets from the fan-in cone of all failing compare points using the the **-failing** option. You can use the **-reference** and **-implementation** options to include only reference or implementation nets respectively.

EXAMPLES

The following example finds nets in the fan-in cones of two compare points,

```
fm_shell (match)> find_region_of_nets {r:/WORK/m1/b1/r1 r:/WORK/m1/b2/r1}
```

The following example finds nets in the top level reference design:

```
fm_shell (match)> find_region_of_nets -cells $ref
```

The following example finds implementation nets in the fan-in cones of failing points:

```
fm_shell (match)> find_region_of_nets -failing -implementation
```

SEE ALSO

```
find_equivalent_nets(2)
```

find_segments

Returns a list of all segments, across the hierarchy of the design, for the given net.

SYNTAX

```
status find_segments
   net_name
```

Data Types

```
net_name string
```

Enabled Shell Modes

Setup

RETURN VALUE

The **find_segments** command returns a list of segments, each specified by absolute path, of the given net, or 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

net_name

Lists segments of the specified net. Specify this must be specified as either an absolute or relative (to **current_instance**) instance-based path.

DESCRIPTION

This command returns a list of segments for the specified net, each specified as an absolute path.

EXAMPLES

The following example shows how to find the net segments of the net: r:/WORK/top/M1/B1/o1 using an absolute path to o1.

```
fm_shell (setup)> find_segments r:/WORK/top/M1/B1/o1
r:/WORK/top/M1/B1/o1 r:/WORK/top/M1/n1

The following example shows how to find the same segments using the current_instance command.
fm_shell (setup)> current_instance r:/WORK/top/M1/B1
fm_shell (setup)> find_segments o1
r:/WORK/top/M1/B1/o1 r:/WORK/top/M1/n1
```

SEE ALSO

```
current_instance(2)
create_net(2)
connect_net(2)
```

find_svf_operation

Finds automated setup (SVF) operation ID numbers based on either command name or operation status, or both.

SYNTAX

```
find_svf_operation
   [ -command command_name]
   [ -status status_name ]
   [ compare_points_list ]
```

Data Types

```
command_name string
status_name string
compare_points_list string
```

ARGUMENTS

-command *command_name*

Specifies the name of the guide command to search for. You need not specify the **guide_** prefix of the command name. Repeat this option to search for multiple commands. Use the following **guide_transformation** sub-types:

- *map*
- *merge*
- *share*
- *tree*

-status *status_name*

Specifies the name of a status to search for. The list returned includes only ID numbers for commands that have the specified status. The option might be repeated to search for multiple statuses. Specify one of the following values for the *status_name* argument:

- *unprocessed* - The operation is read but not yet processed.

- *accepted* - The operation has successfully processed and is applied.
- *rejected* - The processing of the operation failed and the operation is not applied.
- *unsupported* - The tool does not support the operation.
- *unaccepted* - Any status other than accepted.

compare_points_list

Specifies a compare point or a list of compare points. The command searches the fanin of the specified compare points and finds all automated setup (SVF) operations that match by cell name. Use regular expressions to specify points.

DESCRIPTION

Use this command to generate a Tcl list of automated setup (SVF) operation IDs. You can use the list as the argument with the **report_svf_operation** command.

You can optionally specify a list of points and find SVF operations in their fanin cone.

EXAMPLES

```
fm_shell (match)> find_svf_operation -status rejected
3 9 10 11 12

fm_shell (setup)> find_svf_operation -command map
4 5 6 7

fm_shell (setup)> find_svf_operation $ref/U1/point1
5 8

fm_shell (setup)> find_svf_operation -command transformation
3 4 5 6 7

fm_shell (setup)> find_svf_operation -command uniquify
2 8

fm_shell (verify)> report_svf_operation -summary [find_svf_operation -status rejected ]
Operation     Line  Command               Status
-----------------------------------------------------
        3       22  transformation_tree   rejected
        9       95  change_names          rejected
       10      114  change_names          rejected
       11      133  change_names          rejected
       12      153  change_names          rejected
1

fm_shell (verify)> report_svf_operation [ find_svf_operation -status rejected -command tree ]
```

```
SVF Operation 3 (Line: 22) - transformation_tree.  Status: rejected
## Operation Id: 3
guide_transformation \
  -design { test } \
  -type { tree } \
  -input { 16 src1 } \
  -input { 16 src2 } \
  -input { 16 src4 } \
  -input { 16 src6 } \
  -input { 16 src8 } \
  -output { 18 O1 } \
  -pre_resource { { 17 } add_5 = UADD { { src1 ZERO 17 } { src2 ZERO 17 } } } \
  -pre_resource { { 18 } add_5_2 = UADD { { add_5 ZERO 18 } { src4 ZERO 18 } } } \
  -pre_resource { { 18 } add_5_3 = UADD { { add_5_2 } { src6 ZERO 18 } } } \
  -pre_resource { { 18 } sub_5 = USUB { { add_5_3 } { src8 ZERO 18 } } } \
  -pre_assign { O1 = { sub_5 } } \
  -post_resource { { 18 } add_3_root_sub_5 = UADD { { src1 ZERO 18 } { src2 ZERO 18 } } } \
  -post_resource { { 18 } add_2_root_sub_5 = UADD { { src4 ZERO 18 } { src6 ZERO 18 } } } \
  -post_resource { { 18 } add_1_root_sub_5 = UADD { { add_3_root_sub_5 } { add_2_root_sub_5 } } } \
  -post_resource { { 18 } sub_0_root_sub_5 = USUB { { add_1_root_sub_5 } { src8 ZERO 18 } } } \
  -post_assign { O1 = { sub_0_root_sub_5 } }

Info:  guide_transformation 3 (Line: 22)  Could not find pre_resource 'sub_5' in design 'test'.

1

fm_shell (setup)> find_svf_operation -command datapath -status rejected $ref/U1/HardPoints*
14 24 31
```

SEE ALSO

```
remove_guidance(2)
report_svf_operation(2)
report_guidance(2)
set_svf(2)
```

foreach_in_collection

Iterates over the elements of a collection.

SYNTAX

```
string foreach_in_collection
   itr_var
   collections
   body
```

Data Types

```
itr_var            string
collections        list
body               string
```

ARGUMENTS

itr_var

Specifies the name of the iterator variable.

collections

Specifies a list of collections over which to iterate.

body

Specifies a script to execute per iteration.

DESCRIPTION

The **foreach_in_collection** command is used to iterate over each element in a collection. You cannot use the Tcl-supplied **foreach** command to iterate over collections because the **foreach** command requires a list, and a collection is not a list.

The arguments for the **foreach_in_collection** command parallel those of the **foreach** command: an iterator variable, the collections over which to iterate, and the script to apply at each iteration. All arguments are required.

Note: The **foreach_in_collection** command does not allow a list of iterator variables.

During each iteration, the *itr_var* option is set to a collection of exactly one object. Any command that accepts *collections* as an argument also accepts *itr_var* because they are of the same data type (collection).

You can nest the **foreach_in_collection** command within other control structures, including another **foreach_in_collection** command.

Note that if the body of the iteration is modifying the netlist, it is possible that all or part of the collection involved in the iteration will be deleted. The **foreach_in_collection** command is safe for such operations. If a command in the body causes the collection to be removed, at the next iteration, the iteration ends with a message indicating that the iteration ended prematurely.

An alternative to collection iteration is to use complex filtering to create a collection that includes only the desired elements, then apply one or more commands to that collection. If the order of operations does not matter, the following are equivalent. The first is an example without iterators.

```
set s [get_cells {U1/*}]
command1 $s
command2 $s
unset s
```

The following is a similar approach using the **foreach_in_collection** command:

```
foreach_in_collection itr [get_cells {U1/*}] {
  command1 $itr
  command2 $itr
}
```

For collections with large numbers of objects, the non-iterator version is more efficient, though both produce the same results if the commands are order-independent.

EXAMPLES

The following example from Formality sets constants on the output ports of the current instance by using a filter with the **get_ports** command.

```
prompt> set op [get_ports -filter {direction == "out"}]
prompt> set_constant $op 0
Set 'r:/WORK/top/M1/o1' to constant 0
Set 'r:/WORK/top/M1/o2' to constant 0
```

The following example from Formality does the same as above, but using iteration and the Tcl **if** command to do the filtering.

```
prompt> foreach_in_collection itr [get_ports] {
?         if {[get_attribute $itr direction] == "out"} {
?           set_constant $itr 0
?         }
```

```
      ?       }
      Set 'r:/WORK/top/M1/o1' to constant 0
      Set 'r:/WORK/top/M1/o2' to constant 0
```

SEE ALSO

```
collections(2)
foreach(2)
get_ports(2)
set_constant(2)
```

generate_eco_map_file

Generates *guide_eco_change* commands for an RTL that has been modified for ECO.

SYNTAX

```
generate_eco_map_file
   [ -replace ] file
   [ -uncomment_single_operator_mappings ]
```

Data Types

```
file string
```

ARGUMENTS

-replace

Specifies that the file specified by the *file* argument replaces an exisiting ECO map file.

-uncomment_single_operator_mappings

Automatically map operators when only a single mapping option is available.

file

Specifies the name of the file in which to save the ECO map information. Supply the full pathname or only the file name. If you do not specify the directory information, the tool writes information to the current working directory.

DESCRIPTION

Use this command to generate *guide_eco_map* commands for an ECO modification.

Before issuing this command, you must load the original design as the reference, the ECO-modified design as the implementation, and both the original automated setup file (SVF) from synthesis and the *guide_eco_change* automated setup commands that are generated by using the *fm_eco_to_svf* script.

You must manually inspect the resulting SVF file and uncomment the correct mappings. In the case of -*uncomment_single_operator_mappings*, Formality automatically uncomments mappings if there is only one mapping choice. You should still validate that this is the desired behavior. It is possible for an operator to not be mapped to anything.

Map the operators if they perform the same function, have the same type, and have the same size inputs and outputs.

EXAMPLES

The following example writes the ECO map information to the file named *eco_map.svf* in the current working directory.

```
fm_shell> generate_eco_map_file -replace eco_map.svf
Info: wrote file 'eco_map.svf'
```

SEE ALSO

```
guide_eco_map(2)
guide_eco_change(2)
```

get_app_var

Gets the value of an application variable.

SYNTAX

```
string get_app_var
   [-default | -details | -list]
   [-only_changed_vars]
   var
```

Data Types

```
var     string
```

ARGUMENTS

-default

Gets the default value.

-details

Gets additional variable information.

-list

Returns a list of variables matching the pattern. When this option is used, then the *var* argument is interpreted as a pattern instead of a variable name.

-only_changed_vars

Returns only the variables matching the pattern that are not set to their default values, when specified with **-list**.

var

Specifies the application variable to get.

DESCRIPTION

The **get_app_var** command returns the value of an application variable.

There are four legal forms for this command:

-
  ```
  get_app_var <var>
  Returns the current value of the variable.
  ```

-
  ```
  get_app_var <var> -default
  Returns the default value of the variable.
  ```

-
  ```
  get_app_var <var> -details
  ```

Returns more detailed information about the variable. See below for details.

- get_app_var -list [-only_changed_vars] <*pattern*>

Returns a list of variables matching the pattern. If *-only_changed_vars* is specified, then only variables that are changed from their default values are returned.

In all cases, if the specified variable is not an application variable, then a Tcl error is returned, unless the application variable **sh_allow_tcl_with_set_app_var** is set to true. See the **sh_allow_tcl_with_set_app_var** man page for details.

When **-details** is specified, the return value is a Tcl list that is suitable as input to the Tcl **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key, and each even-numbered element in the list is the value of the previous key.

The supported keys are as follows:

name

This key contains the name of the variable. This key is always present.

value

This key contains the current value of the variable. This key is always present.

default

This key contains the default value of the variable. This key is always present.

help

This key contains the help string for the variable. This key is always present, but sometimes the value is empty.

type

This key contains the type of the application variable. Legal values of for this key are: string, bool, int, real. This key is always present.

constraint

This key describes additional constraints placed on this variable. Legal values for this key are: none, list, range. This key is always present.

min

This key contains the min value of the application variable. This key is present if the constraint is range. The value of this key may be the empty string, in which case the variable only has a max value constraint.

max

This key contains the max value of the application variable. This key is present if the constraint is "range". The value of this key may be the empty string, in which case the variable only has a min value constraint.

list

This key contains the list of legal values for the application variable. This key is present if the constraint is "list".

EXAMPLES

The following are examples of the **get_app_var** command:

```
prompt> get_app_var sh_enable_page_mode
1

prompt> get_app_var sh_enable_page_mode -default
false

foreach {key val} [get_app_var sh_enable_page_mode -details] {   echo "$key: $val"
}
=>  name: sh_enable_page_mode
    value: 1
    default: false
    help: Displays long reports one page at a time
    type: bool
    constraint: none

prompt> get_app_var -list sh_*message
sh_new_variable_message
```

SEE ALSO

```
report_app_var(2)
set_app_var(2)
write_app_var(2)
```

get_attribute

Retrieves the value of an attribute on an object or on a collection of objects.

SYNTAX

```
string get_attribute
   [-class class_name]
   [-quiet]
   [-value_list]
   object_spec
   attr_name
```

Data Types

```
class_name        string
object_spec       string or collection
attr_name         string
```

ARGUMENTS

-class *class_name*

Specifies the class name of the *object_spec* option, provided the *object_spec* option is a name. The valid values for the *class_name* option are *lib, lib_cell, lib_pin, design, port, cell, pin, and net.*

-quiet

Indicates that any error and warning messages are not reported.

-value_list

Indicates that the return value should be a list, even if there is only a single object specified to retrieve the attribute. Normally, the return value of the **get_attribute** command is a string if there is only a single object, and a list if multiple objects are specified.

object_spec

Specifies an object from which to retrieve the attribute value. The *object_spec* option must be either a collection of one or more objects, or a name which is combined with the *class_name* option to find the object. If the *object_spec* option is a name, you must also use the *-class* option.

attr_name

Specifies the name of the attribute where the value is retrieved.

DESCRIPTION

Retrieves the value of an attribute on an object or on a collection of objects. The object is either a string name of exactly one object, or a collection of one or more objects. If it is a name, the *-class* option is required. The return value is a string or a list of strings if multiple objects are specified to retrieve an attribute.

EXAMPLES

The following example gets the value of the *direction* attribute for all ports whose names begin with OUT:

```
prompt> get_attribute [get_ports OUT*] direction
out out inout out
```

The following example gets all registers, and for each register prints the register's *full_name* attribute and then gets the register's lib cell and prints the lib cell's *library_name* attribute.

```
prompt> foreach_in_collection sel [all_registers] {
?         echo -n "lib cell of '[get_attribute $sel full_name]' "
?         echo "is in library [get_attribute [get_lib_cell -of_object $sel] library_name]"
?       }
lib cell of 'r:/WORK/top/hold_reg' is in library cmos_slow
lib cell of 'r:/WORK/top/M1/frst_reg' is in library cmos_fast
lib cell of 'r:/WORK/top/M1/scnd_reg' is in library cmos_fast
```

SEE ALSO

```
all_registers(2)
collections(2)
foreach_in_collection(2)
get_lib_cell(2)
get_ports(2)
help_attributes(2)
list_attributes(2)
```

get_cells

Creates a collection of cells from the current design relative to the current instance. You can assign these cells to a variable or pass them into another command.

SYNTAX

```
collection get_cells
   [-exact]
   [-hierarchical]
   [-quiet]
   [-filter expression]
   [patterns | -of_objects objects]
```

Data Types

```
expression          string
objects             list
patterns            list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-hierarchical

Searches for cells level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a cell block1/adder, a hierarchical search finds it using "adder".

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the *expression* value. For any cells that match *patterns* or *objects*, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.

-of_objects *objects*

Creates a collection of cells connected to the specified objects. In this case, each object is either a named pin, a pin collection, a named net or a net collection. The *-of_objects* and *patterns* options are mutually exclusive; you can specify only one. In addition, you cannot use the *-hierarchical* option with the *-of_objects* option.

patterns

Matches cell names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type cell. The *patterns* and *-of_objects* options are mutually exclusive; you can specify only one.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of cells in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any cells match the *patterns* or *objects* option and passes the filter (if specified). If no objects match the criteria, an empty string is returned.

You can use the **get_cells** command at the command prompt, or you can nest it as an argument to another command. For example, you can use it in the **query_objects** command. In addition, you can assign the **get_cells** result to a variable.

When issued from the command prompt, the **get_cells** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_cells** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_cells** command as an argument to the **query_objects** command.

For information about collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the cells that begin with "o" and reference an FD2 library cell. Although the output looks like a list, it is not. The output is just a display.

```
prompt> get_cells "o*" -filter "ref_name == FD2"
{r:/WORK/top/o_reg1 r:/WORK/top/o_reg2 r:/WORK/top/o_reg3 r:/WORK/top/o_reg4}
```

The following example shows that, given a collection of pins, you can query the cells connected to those pins.

```
prompt> set pinsel [get_pins o*/CP]
{r:/WORK/top/o_reg1/CP r:/WORK/top/o_reg2/CP}

prompt> query_objects [get_cells -of_objects $pinsel]
{r:/WORK/top/o_reg1 r:/WORK/top/o_reg2}
```

The following example shows that, given a collection of cells, you can set those cells as verify points.

```
prompt> set_verify_points [get_cells R*]
Set verify point 'r:/WORK/bot/R0_reg'
Set verify point 'r:/WORK/bot/R1_reg'
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_pins(2)
query_objects(2)
set_verify_points(2)
collection_result_display_limit(3)
```

get_command_option_values

Queries current or default option values.

SYNTAX

```
get_command_option_values
   [-default | -current]


   -command command_name
```

Data Types

```
command_name      string
```

ARGUMENTS

-default

Gets the default option values, if available.

-current

Gets the current option values, if available.

-command *command_name*

Gets the option values for this command.

DESCRIPTION

This command attempts to query a default or current value for each option (of the command) that has default and/or current-value-tracking enabled. Details of how the option value is queried depend on whether one of the **-current** or **-default** options is specified (see below).

A "Tcl array set compatible" (possibly empty) list of option names and values is returned as the Tcl result. The even-numbered entries in the list are the names of options that were enabled for default-value-tracking or current-value-tracking and had at least one of these values set to a not-undefined value). Each odd-numbered entry in the list is the default or current value of the option name preceding it in the list.

Any options that were not enabled for either default-value-tracking nor current-value-tracking are omitted from the output list. Similarly, options that were enabled for default-value-tracking or current-value-tracking, but for which no (not-undefined) default or current value is set, are omitted from the result list.

If neither **-current** nor **-default** is specified, then for each command option that has either default-value-tracking or current-value-tracking (or both) enabled, the value returned is as follows:

- The current value is returned if current-value-tracking is enabled and a (not-undefined) current value has been set;
- Otherwise the default value is returned if default-value-tracking is enabled and a (not-undefined) default value has been set;
- Otherwise the name and value pair for the option is not included in the result list.

If **-current** is specified, the value returned for an option is the current value if current-value-tracking is enabled, and a (not-undefined) current value has been set; otherwise the name and value pair for the option is omitted from the result list.

If **-default** is specified, the value returned for an option is the default value if default-value-tracking is enabled, and a (not-undefined) default value has been set; otherwise the name and value pair for the option is omitted from the result list.

The result list from **get_command_option_values** includes option values of both dash options and positional options (assuming that both kinds of options of a command have been enabled for value-tracking).

The command issues a Tcl error in a variety of situations, such as if an invalid command name was passed in with **-command**.

EXAMPLES

The following example shows the use of **get_command_option_values**:

```
prompt> test -opt1 10 -opt2 20
1

prompt> get_command_option_values -command test
-bar1 10 -bar2 20
```

SEE ALSO

```
preview(2)
set_command_option_value(2)
```

get_defined_commands

Get information on defined commands and groups.

SYNTAX

```
string get_defined_commands [-details]
   [-groups]
   [pattern]

string pattern
```

ARGUMENTS

-details

Get detailed information on specific command or group.

-groups

Search groups rather than commands

pattern

Return commands or groups matching pattern. The default value of this argument is "*".

DESCRIPTION

The **get_defined_commands** gets information about defined commands and command groups. By default the command returns a list of commands that match the specified pattern.

When **-details** is specified, the return value is a list that is suitable as input to the **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key name, and each even-numbered element in the list os the value of the previous key. The **-details** option is only legal if the pattern matches exactly one command or group.

When **-group** is specified with **-details**, the supported keys are as follows:

name

This key contains the name of the group.

info

This key contains the short help for the group.

commands

This key contains the commands in the group

When **-details** is used with a command, the supported keys are as follows:

name

This key contains the name of the command.

info

This key contains the short help for the command.

groups

This key contains the group names that this command belongs to.

options

This key contains the options defined for the command. The value is a list.

return

This key contains the return type for the command.

Each element in the *options* list also follows the key value pattern. The set of available keys for options are as follows:

name

This key contains the name of the option.

info

This key contains the short help for the option.

value_info

This key contains the short help string for the value

type

The type of the option.

required

The value will be 0 or 1 depending if the option is optional or required.

is_list

Will be 1 if the option requires a list.

list_length

If a list contains the list length constraint. One of any, even, odd, non_empty or a number of elements.

allowed_values

The allowed values if the option has specified allowed values.

min_value

The minimum allowed value if the option has specified one.

max_value

The maximum allowed value if the option has specified one.

EXAMPLES

```
prompt> get_defined_commands *collection
add_to_collection append_to_collection copy_collection filter_collection
foreach_in_collection index_collection sort_collection
prompt> get_defined_commands -details sort_collection
name sort_collection info {Create a sorted copy of the collection}
groups {} options {{name -descending info {Sort in descending order}
value_info {} type boolean required 1 is_list 0} {name -dictionary info
{Sort strings dictionary order.} value_info {} type boolean required 1
is_list 0} {name collection info {Collection to sort} value_info
collection type string required 0 is_list 0} {name criteria info {Sort
criteria - list of attributes} value_info criteria type list required 0
is_list 1 list_length non_empty}}
```

SEE ALSO

```
help(2)
man(2)
```

get_designs

Creates a collection of designs. You can assign these designs to a variable or pass them into another command.

SYNTAX

```
collection get_designs
   [-exact]
   [-hierarchical]
   [-quiet]
   [-filter expression]
   patterns
```

Data Types

```
expression           string
patterns             list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-hierarchical

Searches for designs inferred by the design hierarchy relative to the current instance. The full name of the object at a particular level must match the patterns.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the *expression* value. For any designs that match *patterns*, the expression is evaluated based on the design's attributes. If the expression evaluates to true, the design is included in the result.

patterns

Matches design names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type design.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of designs that match certain criteria. The command returns a collection if any designs match the *patterns* option and pass the filter, if specified. If no objects matched your criteria, an empty string is returned.

You can use the **get_designs** command at the command prompt, or you can nest it as an argument to another command. For example, you can use it in the **query_objects** command. In addition, you can assign the **get_designs** result to a variable.

When issued from the command prompt, the **get_designs** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable .

The "implicit query" property of the **get_designs** command provides a fast, simple way to display designs in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_designs** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the designs that begin with 'mpu.' Although the output looks like a list, it is just a display.

```
prompt> get_designs mpu*
{r:/WORK/top/mpu_0_0 r:/WORK/top/mpu_0_1 r:/WORK/top/mpu_1_0 r:/WORK/top/mpu_1_1}
```

The following example shows that, given a collection of designs, you can mark those designs as black boxes.

```
prompt> set_black_box [get_designs mpu*]
Set black box on 'r:/WORK/mpu_0_0'
Set black box on 'r:/WORK/mpu_0_1'
Set black box on 'r:/WORK/mpu_1_0'
Set black box on 'r:/WORK/mpu_1_1'
```

SEE ALSO

```
collections(2)
filter_collection(2)
query_objects(2)
set_black_box(2)
collection_result_display_limit(3)
```

get_lib_cells

Creates a collection of library cells. You can assign these library cells to a variable or pass them into another command.

SYNTAX

```
collection get_lib_cells
   [-exact]
   [-quiet]
   [-filter expression]
   patterns | -of_objects objects
```

Data Types

```
expression          string
objects             list
patterns            list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the *expression* value. For any library cells that match the *patterns* or *objects* argument, the expression is evaluated based on the library cell's attributes. If the expression evaluates to true, the library cell is included in the result.

-of_objects *objects*

Creates a collection of library cells that are referenced by the specified cells or own the specified library pins. In this case, each object is either a named library pin or netlist cell, or a library pin collection or a netlist cell collection. The *-of_objects* and *patterns* options are mutually exclusive; you must specify one, but not both.

patterns

Matches library cell names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type lib_cell. The *patterns* and *-of_objects* options are mutually exclusive; you must specify one, but not both.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

This command creates a collection of library cells that match certain criteria. The command returns a collection if any library cells match the *patterns* or *objects* option and pass the filter, if specified. If no objects match the criteria, an empty string is returned.

You can use the **get_lib_cells** command at the command prompt, or you can nest it as an argument to another command. For example, you can use it in the **query_objects** command. In addition, you can assign the **get_lib_cells** command result to a variable.

When issued from the command prompt, the **get_lib_cells** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_lib_cells** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_lib_cells** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries all library cells that are in the misc_cmos library and begin with AN2. Although the output looks like a list, it is just a display.

```
prompt> get_lib_cells /misc_cmos/AN2*
{/misc_cmos/AN2 /misc_cmos/AN2P}
```

The following example shows one way to find out the library cell used by a particular cell.

```
prompt> get_lib_cells -of_objects [get_cells o_reg1]
{r:/misc_cmos/FD2}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_cells(2)
query_objects(2)
collection_result_display_limit(3)
```

get_lib_pins

Creates a collection of library cell pins. You can assign these library cell pins to a variable or pass them into another command.

SYNTAX

```
collection get_lib_pins
   [-exact]
   [-quiet]
   [-filter expression]
   patterns | -of_objects objects
```

Data Types

```
   expression      string
   objects         list
   patterns        list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the *expression* option. For any library cell pins that match the *patterns* option (or the *objects* option), the expression is evaluated based on the library cell pin's attributes. If the expression evaluates to true, the library pin is included in the result.

-of_objects *objects*

Creates a collection of library cell pins referenced by the specified netlist pins or owned by the specified library cells. In this case, each object is either a named library cell or netlist pin, or a library cell collection or netlist pin collection. The *-of_objects* and *patterns* options are mutually exclusive; you must specify one, but not both.

patterns

Matches library cell pin names against patterns. The *patterns* option can include the wildcard characters "*" and "?". The *patterns* option can also include collections of type **lib_pin**. The *patterns* and *-of_objects* options are mutually exclusive; you must specify one, but not both.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

The **get_lib_pins** command creates a collection of library cell pins that match certain criteria. The command returns a collection if any library cell pins match the *patterns* or *objects* options and pass the filter (if specified). If no objects matched the criteria, the empty string is returned.

You can use the **get_lib_pins** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_lib_pins** command result to a variable.

When issued from the command prompt, the **get_lib_pins** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_lib_pins** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_lib_pins** command as an argument to the **query_objects** command. For example, use this if you want to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries all pins of the AN2 library cell in the misc_cmos library. Although the output looks like a list, it is just a display.

```
prompt> get_lib_pins /misc_cmos/AN2/*
{/misc_cmos/AN2/A /misc_cmos/AN2/B /misc_cmos/AN2/Z}
```

The following example shows one way to find out how the library pin is used by a particular pin in the netlist.

```
prompt> get_lib_pins -of_objects o_reg1/Q
{r:/misc_cmos/FD2/Q}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_libs(2)
get_lib_cells(2)
query_objects(2)
collection_result_display_limit(3)
```

get_libs

Creates a collection of libraries. You can assign these libraries to a variable or pass them into another command.

SYNTAX

```
collection get_libs
   [-exact]
   [-quiet]
   [-filter expression]
   [patterns | -of_objects objects]
```

Data Types

```
expression        string
objects           list
patterns          list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the *expression* option. For any libraries that match *patterns* (or *objects*), the expression is evaluated based on the library's attributes. If the expression evaluates to true, the library is included in the result.

-of_objects *objects*

Creates a collection of libraries that contain the specified objects. In this case, each object is either a named library cell or a library cell collection. the *-of_objects* and *patterns* options are mutually exclusive; you can specify only one.

patterns

Matches library names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type lib. The *patterns* and *-of_objects* options are mutually exclusive; you can specify only one.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

The **get_libs** command creates a collection of libraries that match certain criteria. The command returns a collection if any libraries match the *patterns* option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_libs** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_libs** command result to a variable.

When issued from the command prompt, the **get_libs** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_libs** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_libs** command as an argument to the **query_objects** command. For example, use this if you want to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries all loaded libraries. Although the output looks like a list, it is just a display.

```
prompt> get_libs
{r:/misc_cmos r:/misc_cmos_io}
```

The following example shows that given a library collection, you can remove those libraries. Note that you cannot remove libraries if they are referenced by a design.

```
pt_shell> remove_library [get_libs misc*]
Removed technology library 'misc_cmos' from container 'r'
Removed technology library 'misc_cmos_io' from container 'r'
```

SEE ALSO

```
collections(2)
filter_collection(2)
query_objects(2)
remove_library(2)
collection_result_display_limit(3)
```

get_license

Obtains a license for the specified Formality feature(s).

SYNTAX

```
status get_license
   feature_list
```

Data Types

```
feature_list list
```

ARGUMENTS

feature_list

Specifies the list of Formality features to be obtained. The *feature_list* argument might consist of a single value or a space-delimited list of values enclosed within braces ({}).

By looking at your key file, you can determine all of the features licensed at your site.

RETURN VALUE

The **get_license** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command obtains a license for the specified feature(s). These features are checked out by the current user until the **remove_license** command is used or until the program exits.

When **fm_shell** is invoked, Formality will automatically checkout the base Formality license. If this license is unavailable, and the user has set environment variable **FM_WAIT_LICENSE** to 1, then the Formality run will be queued to start as soon as the license becomes available. If **fm_shell -checkout Formality-Ultra** is invoked, Formality will checkout a Formality-Ultra license, or wait for it if **FM_WAIT_LICENSE** is set to 1. The Formality-Ultra license may also be checked out on **fm_shell** startup if environment variable **FM_USE_FORMALITY_ADVANCED_LICENSE** is set to 1.

The **list_licenses** command provides a list of the features that you are currently using.

The **license_users** command provides a list of all features checked out by all users.

EXAMPLES

The following example obtains a Formality-Ultra license from within **fm_shell**:

```
fm_shell (setup)> get_license  Formality-Ultra
```

The following examples show two different ways to obtain a Formality and Formality-Ultra license on **fm_shell** startup:

```
setenv FM_USE_FORMALITY_ADVANCED_LICENSE 1
fm_shell

fm_shell -checkout Formality-Ultra
```

SEE ALSO

```
check_license(2)
license_users(2)
list_licenses(2)
remove_license(2)
```

get_message_ids

Get application message ids

SYNTAX

```
string get_message_ids [-type severity]
[pattern]

string severity
string pattern
```

ARGUMENTS

-type severity

Filter ids based on type (Values: Info, Warning, Error, Severe, Fatal)

pattern

Get IDs matching pattern (default: *)

DESCRIPTION

The **get_message_ids** command retrieves the error, warning and informational messages used by the application. The result of this command is a Tcl formatted list of all message ids. Information about the id can be queried with the **get_message_info** command.

EXAMPLES

The following code finds all error messages and makes the application stop script execution when one of these messages is encountered.

```
foreach id [get_message_ids -type Error] {
  set_message_info -stop_on -id $id
}
```

SEE ALSO

```
print_message_info(2)
set_message_info(2)
suppress_message(2)
```

get_message_info

Returns information about diagnostic messages.

SYNTAX

```
integer get_message_info
   [-error_count | -warning_count | -info_count
     | -limit l_id | -occurrences o_id | -suppressed s_id | -id i_id]
```

Data Types

```
l_id      string
o_id      string
s_id      string
i_id        string
```

ARGUMENTS

-error_count

Returns the number of error messages issued so far.

-warning_count

Returns the number of warning messages issued so far.

-info_count

Returns the number of informational messages issued so far.

-limit *l_id*

Returns the current user-specified limit for a given message ID. The limit was set with the **set_message_info** command.

-occurrences *o_id*

Returns the number of occurrences of a given message ID.

-suppressed *s_id*

Returns the number of times a message was suppressed either using **suppress_message** or due to exceeding a user-specified limit.

-id *i_id*

Returns information about the specified message. The information is returned as a Tcl list compatible with the array set command.

DESCRIPTION

The **get_message_info** command retrieves information about error, warning, and informational messages. For example, if the following message is generated, information about it is recorded:

Error: unknown command 'wrong_command' (CMD-005)

It is useful to be able to retrieve recorded information about generated diagnostic messages. For example, you can stop a script after a certain number of errors have occurred, or monitor the number of messages issued by a single command.

You can also find out how many times a specific message has occurred, or how many times it has been suppressed. Also, you can find out if a limit has been set for a particular message ID.

EXAMPLES

The following example uses the **get_message_info** command to count the number of errors that occurred during execution of a specific command, and to return from the procedure if the error count exceeds a given amount:

```
prompt> proc \
  do_command {limit} {
    set current_errors [get_message_info -error_count]
    command
    set new_errors [get_message_info -error_count]
    if {[expr $new_errors - $current_errors] > $limit} {
      return -code error "Too many errors"
    }
    ...
  }
```

The following example uses the **get_message_info** command to retrieve information on the CMD-014 message:

```
prompt> get_message_info -id CMD-014
id CMD-014 severity Error limit 0 occurrences 0 suppressed 0 message
{Invalid %s value '%s' in list.}
```

SEE ALSO

```
print_message_info(2)
set_message_info(2)
suppress_message(2)
get_message_ids(2)
```

get_nets

Creates a collection of nets from the current design relative to the current instance. You can assign these nets to a variable or pass them into another command.

SYNTAX

```
collection get_nets
   [-boundary_type upper | lower | both]
   [-exact]
   [-hierarchical]
   [-quiet]
   [-filter expression]
   [patterns | -of_objects objects]
```

Data Types

```
expression           string
objects              list
patterns             list
```

ARGUMENTS

-boundary_type upper | lower | both

Specifies what to do when getting nets of boundary pins and ports.

You must specify one of the following values:

- *upper* - Get the net outside the hierarchical block.
- *lower* - Get the net inside the hierarchical block.
- *both* - Get the nets both inside and outside the hierarchical block.

If the *-of_objects* argument is a pin collection or a name that can be either a pin or a port, *upper* is the default value. Otherwise, if *-of_objects* argument is a port collection, *lower* is the default value.

This option can be used only with the *-of_objects* option. In addition, the specified object must be a pin or a port. This option has no meaning for cells.

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-hierarchical

Searches for nets level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a net block1/muxsel, a hierarchical search would find it using "muxsel." You cannot use the -*hierarchical* option with the *-of_objects* option.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with *expression*. For any nets that match *patterns* (or *objects*), the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the net is included in the result.

-of_objects *objects*

Creates a collection of nets connected to the specified objects. Each object is either a named pin, a pin collection, a port, a port collection, a named cell or cell collection. The *-of_objects* and *patterns* options are mutually exclusive; you can specify only one. In addition, you cannot use the *-hierarchical* option with the *-of_objects* option.

patterns

Matches net names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type net. The *patterns* and *-of_objects* options are mutually exclusive; you can specify only one.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

The **get_nets** command creates a collection of nets in the current design, relative to the current instance that match certain criteria. The command returns a collection if any nets match the *patterns* or *objects* and pass the filter (if specified). If no objects matched the criteria, the empty collection is returned.

You can use the **get_nets** command at the command prompt, or you can nest it as an argument to another command (for example, **query_objects**). In addition, you can assign the **get_nets** result to a variable.

When issued from the command prompt, the **get_nets** command behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of **get_nets** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_nets** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the nets that begin with 'NET' in block 'block1'. Although the output looks like a list, it is just a display.

```
prompt> get_nets block1/NET*
{r:/WORK/block1/NET1QNX r:/WORK/block1/NET2QNX}
```

The following example shows that with a collection of pins, you can query the nets connected to those pins.

```
prompt> set pinsel [get_pins {o_reg1/QN o_reg2/QN}]
{r:/WORK/block1/o_reg1/QN r:/WORK/block1/o_reg2/QN}
prompt> query_objects [get_nets -of_objects $pinsel]
{r:/WORK/block1/NET1QNX r:/WORK/block1/NET2QNX}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_pins(2)
query_objects(2)
collection_result_display_limit(3)
```

get_pins

Creates a collection of pins from the current design relative to the current instance. You can assign these pins to a variable or pass them into another command.

SYNTAX

```
collection get_pins
   [-exact]
   [-hierarchical]
   [-quiet]
   [-filter expression]
   [patterns | -of_objects objects]
```

Data Types

```
expression      string
objects         list
patterns        list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-hierarchical

Searches for pins level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the **find** UNIX command. For example, if there is a pin block1/adder/D[0], a hierarchical search finds it using "adder/D[0]". You cannot use the *-hierarchical* option with the *-of_objects* option.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter expression

Filters the collection with the *expression*. For any pins that match *patterns* (or *objects*), the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

-of_objects *objects*

Creates a collection of pins connected to the specified objects. Each object is a named cell or net, or cell collection or pin collection. The-*of_objects* and *patterns* options are mutually exclusive; you can specify only one. In addition, you cannot use the -*hierarchical* option with the -*of_objects* option.

patterns

Matches pin names against patterns. The *patterns* option can include the wildcard characters "*" and "?". The *patterns* option can also include collections of type pin. The *patterns* and -*of_objects* options are mutually exclusive; you can specify only one.

If you do not specify a pattern, the command uses * (asterisk, treated as */*) as the default pattern.

DESCRIPTION

The **get_pins** command creates a collection of pins in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any pins match the *patterns* or *objects* options and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

When used with the -*of_objects* option, the **get_pins** command searches for pins connected to any cells or nets specified in the *objects*.

You can use the **get_pins** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_pins** command result to a variable.

When issued from the command prompt, the **get_pins** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_pins** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_pins** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the 'CP' pins of cells that begin with 'o'. Although the output looks like a list, it is just a display.

```
prompt> get_pins o*/CP
{r:/WORK/top/o_reg1/CP r:/WORK/top/o_reg2/CP r:/WORK/top/o_reg3/CP
r:/WORK/top/o_reg4/CP}
```

The following example shows that given a collection of cells, you can query the pins connected to those cells.

```
prompt> set csel [get_cells o_reg1]
{r:/WORK/top/o_reg1}
prompt> query_objects [get_pins -of_objects $csel]
{r:/WORK/top/o_reg1/D r:/WORK/top/o_reg1/CP r:/WORK/top/o_reg1/CD
r:/WORK/top/o_reg1/Q r:/WORK/top/o_reg1/QN}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_cells(2)
query_objects(2)
collection_result_display_limit(3)
```

get_ports

Creates a collection of ports from the current design relative to the current instance. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection get_ports
   [-exact]
   [-quiet]
   [-filter expression]
   patterns | -of_objects objects
```

Data Types

```
   expression      string
   objects         list
   patterns        list
```

ARGUMENTS

-exact

Considers wildcards to be plain characters, and does not interpret their meaning as wildcards.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-filter *expression*

Filters the collection with the *expression* option. For any ports that match the *patterns* option, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the cell is included in the result.

-of_objects *objects*

Creates a collection of ports connected to the specified objects. Each object is a named net or a net collection. The *-of_objects* and *patterns* options are mutually exclusive; you can specify only one.

patterns

Matches port names against patterns. Patterns can include the wildcard characters "*" and "?". Patterns can also include collections of type port. The *patterns* and *-of_objects* options are mutually exclusive; you can specify only one.

If you do not specify a pattern, the command uses * (asterisk) as the default pattern.

DESCRIPTION

The **get_ports** command creates a collection of ports in the current design or instance that match certain criteria. The command returns a collection if any ports match the *patterns* option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_ports** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_ports** command result to a variable.

When issued from the command prompt, the **get_ports** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_ports** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_ports** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page. In addition, refer to the **all_inputs** and **all_outputs** man pages, which also create collections of ports.

EXAMPLES

The following example queries all input ports beginning with 'mode'. Although the output looks like a list, it is just a display.

```
prompt> get_ports "mode*" -filter {direction == in}
{r:/WORK/mid/mode[0] r:/WORK/mid/mode[1] r:/WORK/mid/mode[2]}
```

The following example sets constant value 0 on ports beginning with "scan_en".

```
prompt> set_constant [get_ports scan_en*] 0
Set 'r:/WORK/top/scan_en' to constant 0
Set 'r:/WORK/top/scan_en_1' to constant 0
Set 'r:/WORK/top/scan_en_2' to constant 0
```

SEE ALSO

```
all_inputs(2)
all_outputs(2)
collections(2)
filter_collection(2)
query_objects(2)
set_constant(2)
collection_result_display_limit(3)
```

getenv

Returns the value of a system environment variable.

SYNTAX

```
string getenv
   variable_name
```

Data Types

```
variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of the environment variable to be retrieved.

DESCRIPTION

The **getenv** command searches the system environment for the specified *variable_name* and sets the result of the command to the value of the environment variable. If the variable is not defined in the environment, the command returns a Tcl error. The command is catchable.

Environment variables are stored in the **env** Tcl array variable. The **getenv**, **setenv**, and **printenv** environment commands are convenience functions to interact with this array.

The application you are running inherited the initial values for environment variables from its parent process (that is, the shell from which you invoked the application). If you set the variable to a new value using the **setenv** command, you see the new value within the application and within any new child processes you initiate from the application using the **exec** command. However, these new values are not exported to the parent process. Further, if you set an environment variable using the appropriate system command in a shell you invoke using the **exec** command, that value is not reflected in the current application.

See the **set**, **unset**, and **printvar** commands for information about working with non-environment variables.

EXAMPLES

In the following example, **getenv** returns you to your home directory:

```
prompt> set home [getenv "HOME"]
/users/disk1/bill

prompt> cd $home

prompt> pwd
/users/disk1/bill
```

In the following example, **setenv** changes the value of an environment variable:

```
prompt> getenv PRINTER
laser1

prompt> setenv PRINTER "laser3"
laser3

prompt> getenv PRINTER
laser3
```

In the following example, the requested environment variable is not defined. The error message shows that the Tcl variable **env** was indexed with the value UNDEFINED, which resulted in an error. In the second command, **catch** is used to suppress the message.

```
prompt> getenv "UNDEFINED"
Error: can't read "env(UNDEFINED)": no such element in array
        Use error_info for more info. (CMD-013)

prompt> if {[catch {getenv "UNDEFINED"} msg]} {
         setenv UNDEFINED 1
       }
```

SEE ALSO

```
catch(2)
exec(2)
printenv(2)
printvar(2)
set(2)
setenv(2)
unsetenv(2)
unset(2)
```

group

Creates a new level of hierarchy.

SYNTAX

```
group
   cell_list
   -design_name design_name
   [-cell_name cell_name]
```

Data Types

```
cell_list string
design_name string
cell_name string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

cell_list

Specifies a list of cells in the current design to group into a new level of hierarchy. To specify more than one cell, enclose them in braces { }.

-design_name *design_name*

Specifies the name of the design containing the new level of hierarchy. The design name must not exist in the library in which it is to be created.

-cell_name *cell_name*

Specifies the name of the instance of the new design. If this option is not specified, a unique cell name is generated.

DESCRIPTION

This command groups any number of cells or instances in the current design into a new design, creating a new level of hierarchy. The new design name must be defined by using the *-design_name* option. A new cell is created in the current design which references the new design.

Specify the instance name of the new design by using the *-cell_name* option. By default, the following naming style is used:

```
cell{integer}
```

where *integer* is an integer value that ensures a unique name.

Ports in the new design and the nets they are connected to in the current design are similarly named. The direction of each port in the new design is determined for pins on the net according to the following table:

Inside pins	Outside pins	Resulting port direction
IN	OUT	IN
OUT	IN	OUT
IN & OUT	OUT	IN
IN & OUT	IN	OUT
IN	IN & OUT	IN
OUT	IN & OUT	OUT
IN & OUT	IN & OUT	INOUT (bidirectional)

Constants (power, ground) that are directly connected to inside pins are included in the new hierarchy. Truncated and undriven inside pins are not represented by a port of the new design. Equivalent inside pins are represented by just one resulting port.

EXAMPLES

This example groups a specified list of cells and creates a new design with the given design name and generates a unique name for its instance.

```
fm_shell (setup)> group -design_name design_name {u1 u2 u3}
```

SEE ALSO

```
current_design(2)
```

guide

Changes the mode to the guide mode.

SYNTAX

```
guide
   [ -append ]
```

ENABLED SHELL MODES

Guide
Setup

ARGUMENTS

-append

Specifies to append to current svf file.

DESCRIPTION

When you use this command, the tool changes to the guide mode and enables all guide commands. When invoked, the tool is in the setup mode. After executing the **guide** command, it switches to the guide mode.

Use automated setup mode (SVF) commands only when the tool is in the guide mode. The tool cannot change to the guide mode after any design information (other than technology libraries) is read. If the design is read, it must be removed before the tool can change to the guide mode.

The prompt changes to indicate the current mode:

```
fm_shell (guide)>
fm_shell (setup)>
fm_shell (match)>
fm_shell (verify)>
```

The **setup** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

```
fm_shell (setup)> guide
1
fm_shell (guide)>
```

SEE ALSO

```
setup(2)
match(2)
verify(2)
undo_match(2)
```

guide_architecture_db

Associates a file with in the .db format with an architectural implementation.

SYNTAX

```
guide_architecture_db
   [ -file filename ]
   [ libraries ]
```

Data Types

```
   filename string
   libraries string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-file *filename*

Specifies the name of a file in the .db format that contains the associated architecture.

libraries

Specifies files in the .db format that are referenced in the architectural implementation.

DESCRIPTION

Use this guide mode command to associate a file in the .db format with an architecture.

EXAMPLES

```
fm_shell (guide)> guide_architecture_db -file arch.db { gtech }
1
```

SEE ALSO

```
guide(2)
guide_architecture_netlist(2)
```

guide_architecture_netlist

Associate a netlist with an architectural implementation.

SYNTAX

```
guide_architecture_netlist
   [ -file filename ]
   [ libraries ]
```

Data Types

```
filename string
libraries string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-file *filename*

Specifies the name of the netlist containing the associated architecture. If the specified *filename* is a directory, then the directory contains a netlist for each design.

libraries

Specifies files in .db file format that are referenced in the architectural implementation.

DESCRIPTION

Use this guide mode command to associate a netlist format implementation for an architecture.

EXAMPLES

```
fm_shell (guide)> guide_architecture_netlist -file arch.net gtech.db
1
```

SEE ALSO

```
guide(2)
guide_architecture_db(2)
```

guide_boundary

Specifies boundaries of individual arithmetic operators within complex datapath netlists.

SYNTAX

```
guide_boundary
   { -body bodyName }
   { -operand operandList }
   { -column columnList }
   { -resource resourceList }
```

Data Types

```
   bodyName string
   operandList string
   columnList string
   resourceList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-body *bodyName*

Specifies the name of the netlist implementing the arithmetic block.

-operand *operandList*

Specifies a list of intermediate results of arithmetic operators.

-column *columnList*

Specifies a list of column specifications for each intermediate operand.

-resource *resourceList*

Specifies a list of arithmetic operators in the block.

DESCRIPTION

Use this guide mode command to specify boundaries of individual arithmetic operators within complex datapath netlists.

EXAMPLES

```
fm_shell (guide)> guide_boundary \
  -body { dp_sub_9_DP_OP_258_5050_0 } \
  -operand { I1 bin 2 } \
  -operand { I2 bin 2 } \
  -operand { I3 bin 2 } \
  -operand { OP3.out.1 bin 4 } \
  -column { I1 0 { I1[0] } } \
  -column { I1 1 { I1[1] } } \
  -column { I2 0 { I2[0] } } \
  -column { I2 1 { I2[1] } } \
  -column { I3 0 { I3[0] } } \
  -column { I3 1 { I3[1] } } \
  -column { OP3.out.1 0 { O1[0] } } \
  -column { OP3.out.1 1 { O1[1] } } \
  -column { OP3.out.1 2 { O1[2] } } \
  -column { OP3.out.1 3 { O1[3] } } \
  -resource { OP3 { I2 I1 I3 } { OP3.out.1 } }
1
```

SEE ALSO

guide(2)

guide_boundary_netlist

Associates a netlist with implementations of compound arithmetic blocks.

SYNTAX

```
guide_boundary_netlist
   { -file filename }
   { libraries }
```

Data Types

```
filename string
libraries string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-file filename

Specifies the name of the netlist containing the associated architecture. If the specified *filename* is a directory, then the directory contains a netlist for each *design*. The name of the file is \'*design*.b.e\'.

libraries

Specifies files in .db format that are referenced in the architectural implementation.

DESCRIPTION

Use this guide mode command to associate a netlist implementation for implementations of compound arithmetic blocks.

EXAMPLES

```
fm_shell (guide)> guide_boundary_netlist -file arch.net { gtech.db }
1
```

SEE ALSO

```
guide(2)
```

guide_change_names

Changes the name of design objects.

SYNTAX

```
guide_change_names
   -design designName
   [ -instance instanceName ]
   [ changeBlock ]
```

Data Types

```
   designName string
   instanceName string
   changeBlock string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the names.

-instance *instanceName*

Specifies the name of the instance containing the names.

changeBlock

Specifies a space separated list specifying the object type, old, and new names.

DESCRIPTION

Use this guide mode command to change object names.

EXAMPLES

```
fm_shell (guide)> guide_change_names -design test \
    { { cell U1 mycell3 } \
    { cell U2 mycell2 } \
    { cell U3 mycell1 } \
                { port data myd } \
    { port clock myck } \
    { port q myq } }
1
```

SEE ALSO

guide(2)

guide_checkpoint

Identifies a checkpoint design.

SYNTAX

```
integer guide_checkpoint
   -type checkpointId
   -file fileName
   [ -design designName ]
   [ -root rootName ]
```

Data Types

```
   checkpointId string
   fileName string
   designName string
   rootName string
```

ENABLED SHELL MODES

guide

ARGUMENTS

-type *checkpointId*

Specifies an identifier tag for a checkpoint.

-file *fileName*

Specifies the name of the sub-directory inside the SVF that contains the checkpoint files.

-root *rootName*

This option is added by Formality when the SVF is written out using **report_guidance** command.

-design *designName*

The name of the design to checkpoint. If omitted, Formality will assume the design is the top design in the reference container.

DESCRIPTION

This guide command identifies a checkpoint.

EXAMPLES

The following example identifies a retiming checkpoint

```
prompt> guide_checkpoint \
    -type { retiming-1 } \
    -file { cp-1 }

1
```

SEE ALSO

```
guide(2)
```

guide_constant

Asserts that a design object can never reach any known value different from a specified constant value.

SYNTAX

```
guide_constant
   -design { designName }
   -verify_scope {  scopeDesignName  }
   -constant0 | -constant1 { designObject  }
```

Data Types

```
   designName string
   scopeDesignName string
   designObject string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design { *designName* }

Specifies the name of the design containing the design object. Specify only the design name, and not the workspace and container names. This design is rewired once the constant guidance is verified and accepted.

-verify_scope { *scopeDesignName* }

Specifies the name of the scope design within which the constant is verifiable.

-constant0 | -constant1 { *designObject* }

Specifies the constant value and the object which can only reach that constant value.

DESCRIPTION

This command asserts that a design object can never reach any known value different from a specified constant value. Formality will verify that the object can in fact load no other value. If that verification succeeds, Formality will drive all loads of the design object with the specified constant.

EXAMPLES

To specify that input port *in0* inside design *test* has been replaced with a constant 0 for all instances of *test* within the scope of top design *top*:

```
fm_shell (guide)> guide_constant -design { test } -verify_scope
     { top } -constant0 { svfObjectPort in0 }
```

To specify that the pin *out0* of instance *I1* is constant 0 in design *test* within the same scope:

```
fm_shell (guide)> guide_constant -design { test  } -verify_scope
     { test } -constant0 { svfObjectPin I1 out0 }
```

SEE ALSO

```
guide(2)
```

guide_constraints

Identifies equivalent and constant block pins.

SYNTAX

```
guide_constraints
   -body bodyName
   [ -const0 { list } ]
   [ -const1 { list } ]
   [ -equivalent { list } ]
```

Data Types

```
bodyName string
list string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-body *bodyName*

Specifies the name of the body tag. This is linked to the instance of the next **guide_datapath** or **guide_multiplier** command with the same bodyName. This is a required option.

-const0 *list*

Wires all pins that are set to a constant 0 if verification against 0 reference succeeds. You can use this option multiple times in the same instance of the command.

-const1 *list*

Wires all pins that are set to a constant 1 if verification against 1 in reference succeeds. You can use this option multiple times in the same instance of the command.

-equivalent *list*

Proves all the specified pins are equivalent or complementary. The proven equivalences are applied to the design. Inverted equivalences are specified with "~" preceding the inverted pin name. You can use this option multiple times in the same instance of the command.

DESCRIPTION

Use this guide mode command to identify arithmetic cell input equivalences and constants to increase the success rate of the arithmetic solver.

EXAMPLES

To set *I1[2]* as equivalent to inverted *I1[3]* in the design *dp* instance *add_22_DP_OP_258_595_1*

```
fm_shell(guide)> guide_constraints -body { dp_add_22_DP_OP_258_595_0 }
     -equivalent { I1[2]  ~ I1[3] }
fm_shell(guide)> guide_datapath       \
  -design { dp }                      \
  -datapath { add_22_DP_OP_258_595_1 }\
  -body { dp_add_22_DP_OP_258_595_0 }
```

SEE ALSO

```
guide(2)
```

guide_datapath

Identifies a datapath subdesign.

SYNTAX

```
integer guide_datapath
   -design designName
   [-instance instanceName]
   -body datapathBody
   [-enable enablePort]
   [-odc_body odcBody]
   [-logic_input { {odcPort} {odcAnchorPoint} }]
```

Data Types

```
   designName string
   instanceName string
   datapathBody string
   enablePort string
   odcBody string
   odcPort string
   odcAnchorPoint string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the datapath.

-instance *instanceName*

Specifies the name of the instance containing the datapath.

-body *datapathBody*

Specifies the name of the file containing the datapath description (.db file or netlist format).

-enable *enablePort*

Specifies the name of the enable port in *datapathBody* specified datapath description.

-odc_body *odcBody*

Specifies the name of the file containing the odc datapath description (netlist format).

-logic_input *odcPort odcAnchorPoint*

Specifies pair of data, first name of the odc port in *odcBody* netlist second name of respective concordant object

DESCRIPTION

This guide mode command identifies a datapath.

EXAMPLES

The following example identifies the datapath for a design named *test* using the datapath description in the file named *dpath.db*.

```
prompt> guide_datapath \
    -design { dp } \
    -datapath { add_22_DP_OP_258_595_1 } \
    -body { dp_add_22_DP_OP_258_595_0 } \
    -enable { DG_ctrl } \
    -odc_body { dp_add_22_DP_OP_258_595_0_odc } \
    -logic_input { { DG_I1 } { top/fifo/controller/pipe_reg[31]/Q } } \
    -logic_input { { DG_I2 } { top/fifo/controller/enable } } \
    -logic_input { { DG_I3 } { top/fifo/sub_controller/sub_465.out.1 31 } }

1
```

SEE ALSO

guide(2)

guide_dont_verify_scan

Identifies a compare point that should be disabled due to scan insertion.

SYNTAX

```
guide_dont_verify_scan
   -ports { port_name ... }
   -pins { pin_name ... }
   -propagate svfTrue
```

Data Types

```
   port_name string
   pin_name string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-ports

Specifies a Tcl list of black box input ports or primary output ports that should not be verified. The format is **design_name**/**port_name**.

-pins

Specifies a Tcl list of black box input pin names that should not be verified. The format is **design_name/cell_name/.../pin_name**.

-propagate svfTrue

Specifies that upstream compare points that do not fanout to other compare points should not be verified. This argument is optional.

DESCRIPTION

This command specifies compare points that should not be verified. This is used when scan circuitry is inserted or scan chains are reordered. This command is ignored if either the **synopsys_auto_setup** or **svf_scan** variables are not set to *true*.

EXAMPLES

To disable the black box pin top/bbox/in1:

```
fm_shell (guide)> guide_dont_verify_scan -pins { top/bbox/in1 }
```

To disable the primary output port top/out1 and propagate dont_verify backward to points that don't fanout elsewhere:

```
fm_shell (guide)> guide_dont_verify_scan -ports { top/out1 } -propagate svfTrue
```

SEE ALSO

```
guide(2)
guide_scan_output(2)
guide_scan_input(2)
synopsys_auto_setup(3)
svf_scan(3)
```

guide_eco_change

Describes changes to lines that are caused by an ECO.

SYNTAX

```
guide_eco_change
   -file_orig { nameOfOriginalFile }
   -file_eco { nameOfEcoFile }
   -type { typeName }
   -original { lineNumber [ lineNumber ] }
   -eco { lineNumber [ lineNumber ] }
```

Data Types

```
   nameOfOriginalFile string
   nameOfEcoFile string
   typeName string
   lineNumber integer
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-file_orig { *nameOfOriginalFile* }

Specifies the name of the original RTL file.

-file_eco { *nameOfEcoFile* }

Specifies th name of the ECO-modified RTL file.

-type { *typeName* }

Specifies the type of operation that was performed on the specified line number. Specify one of the following values for the *typeName* argument:

- `insert for lines added by the ECO-modification.`
- *replace* for lines modified by the ECO-modification.
- *delete* for lines removed by the ECO-modification.

-original { *lineNumber lineNumber* }

Specifies the line number or range of line numbers in the original file.

-eco { *lineNumber lineNumber* }

Specifies the line number or range of line numbers in the ECO-modified file.

DESCRIPTION

Use this command to specify which lines were either inserted, replaced, or deleted as a result of an ECO modification.

Use the *fm_eco_to_svf* script to generate the *guide_eco_change* automated setup commands.

EXAMPLES

```
fm_shell (guide)> guide_eco_change \
  -file_orig { original/rtl/foo.v } \
  -file_eco { eco/rtl/foo.v } \
  -type { insert } \
  -original { 0 } -eco { 1 }

fm_shell (guide)> guide_eco_change \
  -file_orig { original/rtl/foo.v } \
  -file_eco { eco/rtl/foo.v } \
  -type { replace } \
  -original { 4 } -eco { 5 }

fm_shell (guide)> guide_eco_change \
  -file_orig { original/rtl/foo.v } \
  -file_eco { eco/rtl/foo.v } \
  -type { delete } \
  -original { 6 9 } -eco { 6 }

$ cat original/rtl/foo.v
module dummy(A,B,Z1,Z2);
   input  [1:0] A,B;
   output [2:0] Z1,Z2;
   assign       Z1 = A + B;
   assign       Z2 = A - B;
   // Blank
   // Blank
```

```
      // Blank
      // Blank
   endmodule;

   $ cat eco/rtl/foo.v
   // This is the ECO version
   module dummy(A,B,Z1,Z2);
      input  [1:0] A,B;
      output [2:0] Z1,Z2;
      assign       Z1 = A + B + 1;
      assign       Z2 = A - B;
   endmodule;
```

SEE ALSO

```
guide(2)
guide_eco_map(2)
generate_eco_map_file(2)
```

guide_eco_map

Describes the changes to names of arithmetic operators that are caused by an ECO.

SYNTAX

```
guide_eco_map
   -design { designName }
   -from { cellName }
   -to { cellName }
```

Data Types

```
   designName string
   cellName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design { *designName* }

Specifies the name of the design.

-from { *cellName* }

Specifies the name of arithmetic operator when reading the original RTL file.

-to { *cellName* }

Specifies the name of arithmetic operator when reading the ECO-modified RTL file.

DESCRIPTION

Use this guide mode command to specify changed to cell names for lines changed by an ECO modification.

Use the *generate_eco_map_file* command to generate the *guide_eco_map* automated setup commands.

EXAMPLES

```
fm_shell (guide)> guide_eco_map \
  -design { dummy } \
  -from { add_4 } \
  -to { add_5 }

$ cat original/rtl/foo.v
module dummy(A,B,Z1,Z2);
   input  [1:0] A,B;
   output [2:0] Z1,Z2;
   assign       Z1 = A + B;
   assign       Z2 = A - B;
   // Blank
   // Blank
   // Blank
   // Blank
endmodule;

$ cat eco/rtl/foo.v
// This is the ECO version
module dummy(A,B,Z1,Z2);
   input  [1:0] A,B;
   output [2:0] Z1,Z2;
   assign       Z1 = A + B + 1;
   assign       Z2 = A - B;
endmodule;
```

SEE ALSO

guide(2)
guide_eco_change(2)
generate_eco_map_file(2)

guide_environment

Sets the environment variables.

SYNTAX

```
guide_environment
   {
    { name_1 value_1 }
    { name_2 value_2 }
    { name_n value_n }
   }
```

ENABLED SHELL MODES

Guide

ARGUMENTS

{ *name value* }

Specifies a list of variable names and values.

DESCRIPTION

Use this guide mode command to set global environment variables.

EXAMPLES

```
fm_shell (guide)> guide_environment { \
                    { bus_dimension_separator_style ][ } \
                    { bus_extraction_style %s[%d:%d] } \
                    { bus_naming_style %s[%d] } \
                    { bus_range_separator_style : } \
                    { hdlin_while_loop_iterations 1000 } }
1
```

SEE ALSO

guide(2)

guide_fsm_reencoding

Changes the finite state machine (FSM) encoding.

SYNTAX

```
integer guide_fsm_reencoding
   -design design_name
   -previous_state_vector prev_list
   -current_state_vector curr_list
   -state_reencoding state_list
```

Data Types

```
   design_name string
   prev_list string
   curr_list string
   state_list string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *design_name*

Specifies the name of the design containing the FSM.

-previous_state_vector *prev_list*

Specifies a list of register bits before re-encoding.

-current_state_vector *curr_list*

Specifies a list of post re-encoding register bits.

-state_reencoding ***state_list***

Specifies a list of state re-encodings.

DESCRIPTION

This guide mode command changes the FSM encoding.

EXAMPLES

```
prompt> guide_fsm_reencoding \
-design myfsm_0 \
-previous_state_vector { out1_reg out0_reg } \
-current_state_vector { Q4 Q3 Q2 Q1 } \
-state_reencoding { { begin 2#00 2#0001 } \
{ ok    2#01 2#0010 } \
{ nok   2#10 2#0100 } \
{ end   2#11 2#1000 } }
1
```

SEE ALSO

```
guide(2)
```

guide_group

Creates hierarchy around design objects.

SYNTAX

```
integer guide_group
   -design design_name
   [ -instance instance_name ]
   [ -cells cell_list ]
   -new_design new_design_name
   -new_instance new_instance_name
   [group_block]
```

Data Types

```
   design_name string
   instance_name string
   cell_list string
   new_design_name string
   new_instance_name string
   group_block string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *design_name*

Specifies the name of the design containing the objects.

-instance *instance_name*

Specifies the name of the instance containing the objects.

-cells *cell_list*

Specifies a list of the cells being grouped.

-new_design ***new_design_name***

Specifies the name of the new design being created.

-new_instance ***new_instance_name***

Specifies the name of the instance of the new design being created.

group_block

Specifies a space separated list of old-cell and new-cell names.

DESCRIPTION

This guide mode command creates a hierarchy around design objects.

EXAMPLES

```
prompt> guide_group \
-cells { shift4_i_1 \
shift4_i_2 \
shift_4_i_2 \
shift_4_i_3 } \
-design shift8 \
-new_design foo \
-new_instance foo_i_1 \
{ { shift4_i_1 foo_i_1/shift4_i_1 } \
{ shift4_i_2 foo_i_1/shift4_i_2 } }
1
```

SEE ALSO

`guide(2)`

guide_group_function

Instructs HDL readers to create a level of hierachy around specified function call.

SYNTAX

```
guide_group_function
   { -file file_number }
   { -fcall function_call }
   { -new_design new_design_name }
   { -new_instance new_instance_name }
   { -output_name output_port_name }
```

Data Types

```
   file_number number
   function_call string
   new_design_name string
   new_instance_name string
   output_port_name string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-file *file_number*

Specifies the file which includes the function call. File number is assigned in guid_info commands.

-fcall *function_call*

Specifies which function call should be grouped. Naming convention similar to operators.

-new_design *new_design_name*

Specifies the name to call the design representing the function.

-new_instance *new_instance_name*

Specifies the name to call the design representing the function.

-output_name *output_signal_name*

Specifies the name of the output port of the design created for the grouped function.

DESCRIPTION

Use this guide mode command to instruct hdl readers to leave a function call grouped in its own level of hierarchy. This is usefull in isolating XOR trees from surrounding logic.

EXAMPLES

```
fm_shell (guide)> guide_group_function -file 1138  -call func_10_C20
    -new_design foo -new_instance u0 -output_name fee
1
```

SEE ALSO

```
guide(2)
```

guide_hier_map

Specifies a new design name for specially named designs.

SYNTAX

```
guide_instance_map
   -design { designName }
   -cells { listOfCellNames }
   -linked { designName }
```

Data Types

```
   designName string
   listOfCellNames list of strings
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the design containing the cells.

-cells *listOfCellNames*

Lists all cells in the specified design which are instances of the linked design.

-linked *designName*

Specifies the new target name of the listed cells.

DESCRIPTION

Specifies a new design name for specially named designs.

EXAMPLES

```
fm_shell (guide)> guide_instance_map \
                -design { topDesign } \
                -cells { inst1 inst2 } \
                -linked { paramDesign_P1_P2 }
```

SEE ALSO

guide_instance_map(2)

guide_implementation

Identifies the architecture of a DesignWare component.

SYNTAX

```
guide_implementation
   -design designName
   -instance instanceName
   -arch arch
```

Data Types

```
   designName string
   instanceName string
   arch string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the DesignWare component.

-instance *instanceName*

Specifies the name of the instance of the DesignWare component.

-arch *arch*

Specifies the architecture of the DesignWare component. For example *rtl* or *str* for a DW_fp_mult floating point multiplier.

DESCRIPTION

Use this guide mode command to identify the architecture of a DesignWare component.

EXAMPLES

```
fm_shell (guide)> guide_implementation \
                  -design { fp_mul } \
                  -instance { dw_fp_mul } \
                  -arch { str }
1
```

SEE ALSO

```
guide(2)
```

guide_info

Specifies the version of Design Compiler that is used to generate the automated setup file for verification (SVF).

SYNTAX

```
guide_info
   [ -file fileName ]
   [ -version { version } ]
```

Data Types

```
fileName string
version string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-file *fileName*

Specifies the name of the automated setup file for verification (SVF).

-version *version*

Specifies the version of Design Compiler that is used to create the automated setup file that is specified by using the *-file* option.

DESCRIPTION

Specifies the version of Design Compiler that is used to generate the automated setup file. This information is used to control the way the tool names datapath operators.

EXAMPLES

```
fm_shell (guide)> guide_info -version { rtl.v 6.346 }
```

SEE ALSO

guide_instance_map

Specifies a new design name for specially named designs.

SYNTAX

```
guide_instance_map
   -design { designName }
   [ -instance { instanceName } ]
   -linked { designName }
```

Data Types

```
designName string
instanceName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the design containing the instance.

-instance *instanceName*

Specifies the hierarchical path from the specified design to an instance of a parameterized, or a specially named, design.

-linked *designName*

Specifies the new target name of the instance design.

DESCRIPTION

Specifies a new design name for specially named designs.

EXAMPLES

```
fm_shell (guide)> guide_instance_map \
                -design { paramDesign } \
                -instance { inst1 } \
                -linked { paramDesign_P1_P2 }
```

SEE ALSO

guide_instance_merging

Identifies merged instances.

SYNTAX

```
guide_instance_merging
   -design designName
   -from fromList
   -to toInstance
```

Data Types

```
   designName string
   fromList string
   toInstance string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the merged instances.

-from *fromList*

Lists the merged instances.

-to *toReg*

Specifies the final instance.

DESCRIPTION

Use this guide mode command to identify merged instances.

EXAMPLES

```
fm_shell (guide)> guide_instance_merging \
                     -design top \
                     -from { inst3 inst2 } \
                     -to inst1
1
```

SEE ALSO

guide(2)

guide_inv_push

SYNTAX

```
guide_inv_push
   -design { designName }
   -register { registerName }
```

Data Types

```
   designName string
   registerName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the phase inverted (inversion pushed) register.

-register *registerName*

Specifies the name of the phase inverted register.

DESCRIPTION

Use this guide mode command to indicate the named register is phase inverted (inversion pushed).

EXAMPLES

```
fm_shell (guide)> guide_inv_push -design test -register q1_reg[0]
1
```

SEE ALSO

svf_inv_push(3)

guide_invert

Use this guide mode command to indicate that the named object is inverted.

SYNTAX

```
guide_invert
   -design { designName }
   -object { designObject  }
```

Data Types

```
   designName string
   designObject string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design { *designName* }

Specifies the name of the design containing the design object. Specify only the design name, and not the workspace and container names.

-object { *designObject* }

Specifies the name of the object which is inverted.

DESCRIPTION

Formality will insert an inverter in the design and all its up cells at the design object interface. For an input port object that translates to inserting an inverter on the net that drives the input port in all the up cells and inserting an inverter on the net that is driven by the input port inside the design. For an output port, an inverter is inserted on the net that drives the output port and also on all upcell nets driven by the output port.

EXAMPLES

To specify that input port *in0* inside design *test* has been inverted for all instances of *test* within the scope of top design *top*:

```
fm_shell (guide)> guide_invert -design { test } -object { svfObjectPort in0 }
```

SEE ALSO

```
guide(2)
```

guide_mark

SYNTAX

```
guide_mark
   type
   phase
```

Data Types

```
type string
phase string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

type

phase

DESCRIPTION

This command is for internal use only.

guide_mc

Identifies a subdesign generated by Module Compiler.

SYNTAX

```
status guide_mc
   -design designName
   [mcBlock]
```

Data Types

```
designName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design.

mcBlock

Specifies a list of old-name new-name pairs.

DESCRIPTION

This command identifies a subdesign generated by Module Compiler.

EXAMPLES

This example shows how to use the **guide_mc** command.

```
fm_shell (guide)> guide_mc -design top { {from to} {old new} }
1
```

SEE ALSO

```
guide(2)
```

guide_merge

Groups arithmetic operators together.

SYNTAX

```
guide_merge
   { -design designName }
   { -datapath datapathName }
   { -input inputList }
   { -output outputList }
   { -pre_resource preResourceList }
   { -pre_assign preAssignList }
```

Data Types

```
designName string
datapathName string
inputList string
outputList string
preResourceList string
preAssignList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design to be transformed.

-datapath *datapathName*

Specifies the name of the datapath block.

-input *inputList*

Specifies a list of block inputs.

-output ***outputList***

Specifies a list of block outputs.

-pre_resource ***preResourceList***

Specifies a list of resources in the arithmetic block pre-graph.

-pre_assign ***preAssignList***

Specifies a list of assignments in the arithmetic block pre-graph.

DESCRIPTION

Use this guide mode command to group arithmetic operators together into a new datapath block.

EXAMPLES

```
fm_shell (guide)> guide_merge \
  -design { sig_calc } \
  -datapath { sub_113_DP_OP_284_7544_12 } \
  -input { 36 I1 } \
  -input { 15 I2 } \
  -input { 12 I3 } \
  -input { 5 I4 } \
  -input { 1 I5 } \
  -output { 36 O6 } \
  -output { 24 O4 } \
  -pre_resource { { 27 } mult_113 = MULT { { I2 } { I3 } } } \
  -pre_resource { { 36 } sub_113 = SUB { { I1 } { mult_113 SIGN 36 } } } \
  -pre_resource { { 24 } srl_125 = \
    SHIFT { { sub_113 35 12 } { I4 } { U`b0 } { U`b0 } { U`b0 } { U`b0 } } } \
  -pre_resource { { 24 } add_125 = ADD { { srl_125 } { I5 ZERO 24 } } } \
  -pre_assign { O6 = { sub_113 } } \
  -pre_assign { O4 = { add_125 } }
1
```

SEE ALSO

```
guide(2)
```

guide_multibit

Support name-matching of multibit banked registers.

SYNTAX

```
status guide_multibit
     -design designName
     -type [ svfMultibitTypeBank | svfMultibitTypeSplit ]
     -groups { { cellList } ... }
```

Data Types

```
designName string
cellList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the objects.

-type *svfMultibitTypeBank* | *svfMultibitTypeSplit*

Specifies whether this is a banking operation or split operation

-groups {*{cellList}* ...}

For a banking operation (-type *svfMultibitTypeBank*), -groups specifies a list of cellLists that represent registers that are mapped into a bank. Each cellList in the group contains as list of register names and sizes, the last two elements specify the name of the banked register instance and its size.

For a split operation (-type *svfMultibitTypeSplit*), -groups specifies a list of cellLists that represent registers that are split into sub-banks. For each cellList, the first element specifies a banked register name followed by its size and the remaining elements specify the list of sub-bank register instance names and sizes.

DESCRIPTION

Use this guide mode command to enable Formality to name-match registers in the reference to the corresponding banked/split registers in the implementation.

EXAMPLES

```
fm_shell (guide)> guide_multibit \
            -design myDesign \
            -type svfMultibitTypeBank \
            -groups \
            { { i_reg[7:0] 8 j_reg[2:0] 3 k_reg 1 i_reg_7_0_j_reg_2_0_k_reg_bank_reg 12 } \
            { d_reg[12:0] 13 d_reg_12_0_bank_reg 13 } }
1
fm_shell (guide)> guide_multibit \
                     -design myDesign \
                     -type svfMultibitTypeSplit \
                     -groups \
                     { { i_reg_7_0_j_reg_2_0_k_reg_bank_reg 12 \
                         i_reg_7_0_j_reg_2_0_k_reg_bank_reg_bank[6_0] 7 \
                         i_reg_7_0_j_reg_2_0_k_reg_bank_reg_bank[7] 1 \
                         i_reg_7_0_j_reg_2_0_k_reg_bank_reg_bank[11_8] 4 } \
                       { d_reg_12_0_bank_reg 13 \
                         d_reg_12_0_bank_reg_bank[5_0] 6 \
                         d_reg_12_0_bank_reg_bank[12_6] 7} }
1
```

SEE ALSO

```
guide_change_names(2)
guide_group(2)
```

guide_multiplier

Identifies a subdesign as a multiplier with a specific architecture.

SYNTAX

```
guide_multiplier
   -design designName
   [ -instance instanceName ]
   -arch arch
   -body fileName
   [ -rounding { ExtPos [ IntPos ] } ]
   [-enable enablePort]
   [-odc_body odcBody]
   [-logic_input { {odcPort} {odcAnchorPoint} }]
```

Data Types

```
designName string
instanceName string
arch string
fileName string
ExtPos int
IntPos int
enablePort string
odcBody string
odcPort string
odcAnchorPoint string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the multiplier.

-instance ***instanceName***

Specifies the name of the instance containing the multiplier.

-arch ***arch***

Specifies the architecture of the multiplier: *csa*, *pparch*, or *apparch*.

-body ***fileName***

Specifies the file containing the multiplier description, either a .db file or a netlist.

-rounding ***{ ExtPos IntPos }***

Specifies the external and internal rounding positions, if any.

-enable ***enablePort***

Specifies the name of the enable port in *datapathBody* specified datapath description.

-odc_body ***odcBody***

Specifies the name of the file containing the odc datapath description (netlist format).

-logic_input ***odcPort odcAnchorPoint***

Specifies pair of data, first name of the odc port in *odcBody* netlist second name of respective concordant object

DESCRIPTION

Use this guide mode command to identify a subdesign as a multiplier with a specific architecture.

EXAMPLES

```
fm_shell (guide)> guide_multiplier \
                  -design { fei_int } \
                  -instance { mul_24/mult/mult } \
                  -arch { csa } \
                  -body { fei_int_DW02_mult_8_8_0 } \
                  -rounding { 8 6 } \
                  -enable { DG_ctrl } \
                  -odc_body { fei_int_DW02_mult_8_8_0_odc } \
                  -logic_input { { DG_I2 } { enable } }
1
```

SEE ALSO

```
guide(2)
```

guide_port_constant

Identifies an input port that is tied to a constant.

SYNTAX

```
guide_port_constant
   -design design_name
   -value 0 | 1 | X
   -ports { port_name ... }
```

Data Types

```
design_name string
port_name string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *design_name*

Specifies the name of the design whose ports is tied to a constant.

-value *0* | *1* | *X*

Specifies a value (0,1,X) that is applied to the input port as a constant.

-ports *port_name*

Specifies a Tcl list of input port names that is tied to the value specified by the **-value** option.

DESCRIPTION

Use the **guide_port_constant** command to specify the input ports that is held constant. The tool ignores this command if either the **synopsys_auto_setup** variable or the **svf_port_constant** variable is not set to *true*.

EXAMPLES

To set the port en of design top to 1:

```
fm_shell (guide)> guide_port_constant -design top -value 1 -ports { en }
```

SEE ALSO

```
guide(2)
synopsys_auto_setup(3)
svf_port_constant(3)
```

guide_private

Instructs svf processing to obtain and execute the next guidance from an associated private file.

SYNTAX

```
guide_private
   [ -root dir_path ]
   [ -skip ]
```

Data Types

```
dir_path string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-root *dir_path*

Specifies the path to the private file containing the private guidance

-skip

Instructs svf processing to skip, or not process, the private svf guide (in the associated private file) corresponding to this guide_private.

DESCRIPTION

Use this guide mode command to direct svf processing to access and execute the next available guidance command to be found in a private file associated with the (public) svf in which this guide_private command was found.

EXAMPLES

```
fm_shell (guide)> guide_private

fm_shell (guide)> guide_private -skip

fm_shell (guide)> guide_private -root private_svf_2/private

1
```

SEE ALSO

guide(2)

guide_reg_constant

Asserts that a register can never reach any known value different from a specified constant value.

SYNTAX

```
guide_reg_constant
   [ -design designName ]
   [ -replaced svfTrue | svfFalse ]
   instanceName
   constantVal
```

Data Types

```
   designName string
   instanceName string
   constantVal integer
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the register. Specify only the design name, and not the workspace and container names.

-replaced svfTrue | svfFalse

Specifies whether the register has been replaced, in the implementation, with the constant.

instanceName

Specifies the hierachical path from the specified design to the target register.

constantVal

Specifies the constant value.

DESCRIPTION

This command asserts that a register can never reach any known value different from a specified constant value, and to inform Formality whether or not the register has been replaced, in the implementation, with that constant. Formality will verify that the register can in fact load no other value. If that verification succeeds and "-replaced" is "svfTrue", Formality will replace the register with the specified constant. If that verification succeeds and "-replaced" is "svfFalse", Formality will not replace the register but will use the verified constant value as a constraint for subsequent verification. For example, if the constant value is 0, Formality will allow only the values 0 and X, not 1, as previous states for the register in subsequent verification.

If "-replaced" is not specified, Formality will assume the register has not been replaced, unless/until a subsequent SVF pre-verification depending on the register fails, at which point it will re-attempt the verification with the assumption that the register has been replaced. This results in nonoptimal run time, and can eventually lead to false differences if the register has not in fact been replaced. Therefore, though -replaced is optional for backward-compatibility, it is recommended to always specify it.

EXAMPLES

To specify that the instance *U1* inside design *test* has been replaced with a constant 1:

```
fm_shell (guide)> guide_reg_constant -design test U1 1 -replaced svfTrue
```

To specify that the instance *r:WORK/top/mid_inst_0/bot_inst_0/state[0]* can only load constant *0*, but has not been replaced with a constant 0:

```
fm_shell (guide)> guide_reg_constant -design top mid_inst_0/bot_inst_0/state[0] 0 -replaced svfFals
```

SEE ALSO

```
guide(2)
```

guide_reg_duplication

Identifies duplicated registers.

SYNTAX

```
guide_reg_duplication
   [-design designName]
   -from fromReg
   -to toList
```

Data Types

```
   designName string
   fromReg string
   toList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the duplicate registers. By default, the command operates on all designs.

-from *fromReg*

Specifies the name of the original register.

-to *toList*

Specifies a list of duplicated registers.

DESCRIPTION

This guide mode command identifies duplicated registers.

EXAMPLES

The following example shows the registers that are duplicates of the register named *UI* in a design named *test*.

```
prompt> guide_reg_duplication \
-design test \
-from U1 \
-to { U1 U2 U3 }
1
```

SEE ALSO

```
guide(2)
```

guide_reg_encoding

Identifies registers whose encoding has changed, usually from binary to carry-save.

SYNTAX

```
integer guide_reg_encoding
   -design designName
   -from fromStyle
   -to toStyle
   {bit bit [ bit]*}
```

Data Types

```
   designName string
   fromStyle string
   toStyle string
   bit String
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the encoded register.

-from *fromStyle*

Specifies the original style of the register.

-to *toStyle*

Specifies the new style of the register.

bit

Specifies the register bits.

DESCRIPTION

This guide mode command identifies bits of registers that are re-encoded. For a binary to carry-save encoding, the first bit on each line represents the binary value, and the remaining bits represent the carry-save encoding.

EXAMPLES

The following example shows how to identify registers whose encoding has changed from binary to CS2 for a design named *test*.

```
prompt> guide_reg_encoding \
-design test \
-from binary \
-to CS2 \
{ R[0] R_sum[0] R_carry[0] } \
{ R[1] R_sum[1] R_carry[1] }
1
```

SEE ALSO

```
guide(2)
```

guide_reg_eqop

Identifies registers that are equal and opposite. These registers are needed to verify the **guide_reg_constant** and **guide_reg_merging** commands.

SYNTAX

```
status guide_reg_eqop
   [ -design designName ]
   -from FromList
   -to ToReg
```

Data Types

```
designName string
FromList string
ToReg string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the registers that are merged according to the automated setup file.

-from *FromList*

Specifies a list of the equal or opposite registers. Use the "~" symbol to indicate opposite functionality.

-to *ToReg*

Specifies the register to which the pins are merged for preverification.

DESCRIPTION

Use this command to identify registers with equivalent functionality that are used to verify the **guide_reg_merging**, and **guide_reg_constant** commands. This command does not modify the reference design. After the related **guide_reg_merging** and **guide_reg_constant** commands are processed, the tool ignores this command.

EXAMPLES

```
fm_shell (guide)> guide_reg_eqop \
                  -design top \
                  -to { Q2 }\
                  -from { Q2 Q3 ~Q4 } \
1
```

SEE ALSO

```
guide(2)
guide_reg_merging(2)
guide_reg_constant(2)
```

guide_reg_merging

Identifies merged registers.

SYNTAX

```
guide_reg_merging
   [ -design designName ]
   -from fromList
   -to toReg
```

Data Types

```
designName string
fromList string
toReg string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the merged registers.

-from *fromList*

Lists the merged registers.

-to *toReg*

Specifies the final register.

DESCRIPTION

Use this guide mode command to identify merged registers.

EXAMPLES

```
fm_shell (guide)> guide_reg_merging \
                      -design top \
                      -from { Q3 } \
                      -to Q2
1
```

SEE ALSO

guide(2)

guide_reg_removal

Identifies registers that been removed during synthesis.

SYNTAX

```
guide_reg_removal
   -design designName
   -cells cellList
```

Data Types

```
designName string
cellList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the removed registers.

-cells *fromList*

Specifies a list of removed registers.

DESCRIPTION

Use this guide mode command to identify registers which have been removed. NOTE: This automated setup file is for information only and is not to be used or verified by Formality.

EXAMPLES

```
fm_shell (guide)> guide_reg_removal \
             -design top \
             -cells { reg1 reg2 }
1
```

SEE ALSO

guide_reg_split

Describes how to split loads of a given register onto its many duplicate copies.

SYNTAX

```
guide_reg_split
   -design {designName}
   -from {RegName}
   -to {RegName:Pin LoadList}}
```

Data Types

```
designName string
RegName string
Pin string
LoadList TCL list of cell:pin pairs
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the registers.

-from *RegName*

Specifies the original register.

-to *RegName:Pin LoadList*

Specifies the final registers and their load connections. Multiple occurrences of this option are legal.

DESCRIPTION

Use this guide mode command to specify split registers.

EXAMPLES

```
fm_shell (guide)> guide_reg_split \
                      -design { top } \
                      -from { reg1 } \
                      -to { reg11:Q { { C1:A } { C2:B } } } \
                      -to { reg12:Q { { C3:A } } }
1
```

SEE ALSO

```
guide_reg_merging(2)
```

guide_rename_design

Renames a design.

SYNTAX

```
guide_rename_design
   -design oldName
   -new_design newName
```

Data Types

```
   oldName string
   newName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *oldName*

Specifies the current name of the design.

-new_design *newName*

Specifies the new name of the design.

DESCRIPTION

Use this guide mode command to rename a design.

EXAMPLES

```
fm_shell (guide)> guide_rename_design \
                  -design oldName \
                  -new_design newName
1
```

SEE ALSO

```
guide(2)
```

guide_replace

Describes high-level arithmetic optimizations.

SYNTAX

```
guide_replace
   -origin { originName }
   [ -type { typeName } ]
   -design { designName } | -body { bodyName }
   -input { inputList }
   -output { outputList }
   -pre_resource { preResourceList }
   -pre_assign { preAssignList }
   -post_resource { postResourceList }
   -post_assign { postAssignList }
```

Data Types

```
originName string
typeName string
designName wstring
bodyName string
inputList string
outputList string
preResourceList string
preAssignList string
postResourceList string
postAssignList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-origin { *originName* }

Specifies the origin of the optimization.

-type { *typeName* }

Specifies the need for special case processing.

-design { *designName* }

Specifies the design in which the optimization takes place, if it is a design-based optimization.

-body { *bodyName* }

Specifies the netlist implementing the arithmetic block, if it is a block-based optimization.

-input { *inputList* }

Lists the optimization inputs.

-output { *outputList* }

Lists the optimization outputs.

-pre_resource { *preResourceList* }

Lists the resources in the optimization pre-graph.

-pre_assign { *preAssignList* }

Lists the assignments in the optimization pre-graph.

-post_resource { *postResourceList* }

Lists the resources in the optimization post graph.

-post_assign { *postAssignList* }

Lists the assignments in the optimization post graph.

DESCRIPTION

Use this guide mode command to specify high-level arithmetic optimizations in a design or on a datapath block.

EXAMPLES

```
fm_shell (guide)> guide_replace \
  -origin { Gensh } \
  -body { dp_sub_9_DP_OP_258_5050_0 } \
  -input { unsigned 2 I1 } \
  -input { unsigned 2 I2 } \
  -input { unsigned 2 I3 } \
  -output { unsigned 4 O1 } \
  -pre_resource { { 4 } OP0 = MULT { { I1 ZERO 4 } { I1 ZERO 4 } } } \
  -pre_resource { { 4 } OP1 = MULT { { I2 ZERO 4 } { I2 ZERO 4 } } } \
  -pre_resource { { 4 } OP2 = ADD { { OP0.out.1 ZERO 4 } { OP1.out.1 ZERO 4 } } } \
  -pre_resource { { 4 } OP3 = SUB { { OP2.out.1 ZERO 4 } { I3 ZERO 4 } } } \
  -pre_assign { O1 = { OP3.out.1 ZERO 4 } } \
  -post_resource { { 4 } OP3 = SOP { { { I2 ZERO 4 } { I2 ZERO 4 } } { { I1 ZERO 4 }
     { I1 ZERO 4 } } { { - I3 ZERO 4 } } } } \
  -post_assign { O1 = { OP3.out.1 ZERO 4 } }

fm_shell (guide)> guide_replace \
  -origin { Presto_aco } \
  -type { svfReplacePrestoConditionalAccumulation } \
  -design { digit_package } \
  -input { 1 src_1 } \
  -input { 1 src_2 } \
  -input { 10 src_3 } \
  -input { 10 src_4 } \
  -output { 10 aco_out } \
  -pre_resource { { 10 } add_837 = ADD { { src_3 ZERO 10 } { src_4 ZERO 10 } } } \
  -pre_resource { { 10 }  C921 = SELECT { { src_1 } { src_2 } { add_837 ZERO 10 }
     { src_3 ZERO 10 } } } \
  -pre_assign { aco_out = {  C921 ZERO 10 } } \
  -post_resource { { 10 } mult_add_837_aco = MULT { { src_4 ZERO 10 }
     { src_1 ZERO 10 } } } \
  -post_resource { { 10 } add_837_aco = ADD { { src_3 ZERO 10 }
     { mult_add_837_aco ZERO 10 } } } \
  -post_assign { aco_out = { add_837_aco ZERO 10 } }
1
```

SEE ALSO

guide(2)

guide_retiming

Describes a basic register retiming move over a library cell or net fork.

SYNTAX

```
integer guide_retiming
   -design {designName}
   -direction {direction}
   -libCell {cellName}
   -input {cellInput}
   -output {cellOutput}
   [-resetStateDontCare]
```

Data Types

```
   designName string
   direction string
   cellName string
   cellInput string
   cellOutput string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains retimed logic.

-direction *direction*

Specifies the direction that the given register has moved, forward or backward.

-libCell *cellName*

Specifies the name of the library cell across which the register has moved or **FM_FORK** for a net fork.

-input ***cellInput***

Specifies the input to the cell across which the register has moved, in the format *PinName*:*RegisterName*. Multiple occurrences of this option are legal.

-output ***cellOutput***

Specifies the output of the cell across which the register has moved, in the format *PinName*:*RegisterName*. Multiple occurrences of this option are legal.

-resetStateDontCare

Indicates whether the reset state of the register changed during the retiming move.

DESCRIPTION

This guide mode command describes a basic register retiming move over a library cell or net fork.

EXAMPLES

The following example shows the command used with a design named *test* in a *forward* direction on a library cell named *NAND2* with multiple inputs and output as shown.

```
prompt> guide_retiming \
-design {test} \
-direction {forward} \
-libCell {NAND2} \
-input {A:q1_reg} \
-input {B:q2_reg} \
-output {Y:__tmp__name___0}
1
```

SEE ALSO

```
guide_retiming_decompose(2)
guide_retiming_finished(2)
guide_retiming_multibit(2)
svf_retiming(3)
```

guide_retiming_class

Identifies a given SEQ as the leader of the named retiming Class.

SYNTAX

```
guide_retiming_class
   -design {designName}
   -name {ClassName}
   -leader {RegName}
```

Data Types

```
   designName string
   ClassName string
   RegName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the cell in -leader.

-name *ClassName*

Specifies the name of the retiming class.

-leader *RegName*

Specifies the name of the SEQ that is chosen to be the leader of the named retiming class.

DESCRIPTION

This guide mode command identifies a given SEQ as the leader of the named retiming Class. Class is determined by SL, SC, SS, AL, AC, AS, CLK nets of a given SEQ. Two SEQs Reg1 and Reg2 are in the same class if their SL1=SL2, CLK1=CLK2, AL1=AL2, SC1|SS1=SC2|SS2, and AC1|AS1=AC2|AS2.

EXAMPLES

The following example shows the command used with a design named *test* and on register instance "reg1" which is a multibit (4 bit) register.

```
prompt> guide_retiming_class \
-design {test} \
-name {Class_1} \
-leader {REG_0}
```

SEE ALSO

```
guide_retiming_pinmap(2)
guide_retiming_unmap(2)
guide_retiming_seqmap(2)
guide_retiming_cross(2)
guide_retiming_move(2)
```

guide_retiming_cross

Describes a register moving across a hierarchy.

SYNTAX

```
guide_retiming_cross
   -design {designName}
   -direction {Direction}
   -class {ClassName}
   -from {RegName}
   -to {RegName}
```

Data Types

```
   designName string
   Direction string, either svfForward or svfBackward
   ClassName srring
   RegName string, hierarchical register name relative to -design
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies a register move crossing hierarchy.

-direction *Direction*

Specifies the move direction (forward or backward)

-class *ClassName*

Specifies the name of the retiming class.

-from *RegName*

Specifies the name of the existing register (SEQ) before the move. RegName can be hierarchical relative to -design.

-to *RegName*

Specifies the name of the new register (SEQ) after the move. RegName can be hierarchical relative to -design.

DESCRIPTION

This guide mode command specifies a register move across design hierarch.

EXAMPLES

The following example shows the command used with a design named *test* to move *REG_O* across design hierarchy *U1*.

```
prompt> guide_retiming_cross \
-design {test} \
-direction {svfForward} \
-class {Class_1} \
-from {REG_0}
-to {U1/REG_10}
```

SEE ALSO

```
guide_retiming_pinmap(2)
guide_retiming_unmap(2)
guide_retiming_seqmap(2)
guide_retiming_class(2)
guide_retiming_move(2)
```

guide_retiming_decompose

Indicates that a synchronous enable register is decomposed - changed to a D flip-flop with a feedback MUX.

SYNTAX

```
integer guide_retiming_decompose
   -design {designName}
   -preName {previousName}
   -postName {changedName}
   [ -classChanged ]
```

Data Types

```
   designName string
   previousName string
   changedName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the decomposed register.

-preName *previousName*

Specifies the name of the register before decomposition.

-postName *changedName*

Specifies the name of the register after decomposition. If the name is not changed, then the postName is the same as the preName.

-classChanged

Indicates whether the class of the register has changed or not.

DESCRIPTION

Use this guide mode command to indicate that a synchronous enable register has changed to a D flip-flop with a feedback MUX, or decomposed.

EXAMPLES

```
prompt> guide_retiming_decompose \
-design {test} \
-preName {q1_reg[0]} \
-postName {q1_reg[0]} \
-classChanged
1
```

SEE ALSO

```
guide_retiming(2)
guide_retiming_finished(2)
guide_retiming_multibit(2)
svf_retiming(3)
```

guide_retiming_finished

Indicates that the group of automated setup (SVF) retiming commands for this design is finished.

SYNTAX

```
status guide_retiming_finished
```

ENABLED SHELL MODES

Guide

ARGUMENTS

None

DESCRIPTION

This guide mode command indicates that the group of automated setup (SVF) retiming commands for this design is finished.

EXAMPLES

The following example shows that the group of automated setup (SVF) retiming commands for this design is finished.

```
prompt> guide_retiming_finished
1
```

SEE ALSO

```
guide_retiming(2)
guide_retiming_decompose(2)
guide_retiming_multibit(2)
svf_retiming(3)
```

guide_retiming_move

Describes how registers are moved (retimed) across a given gate instance.

SYNTAX

```
guide_retiming_move
   -design {designName}
   -direction {moveDirection}
   -class {className}
   -cell {cellName}
   -input {Pin:regName:AsyncState:SyncState}
   -output {Pin:regName:AsyncState:SyncState}
```

Data Types

```
   designName string
   moveDirection string (either svfForward or svfBackward)
   className string
   cellName string
   Pin string
   regName string
   AsyncState integer (between 0 and 7), or nothing if no async clear/set on this register
   SyncState integer (between 0 and 7), or nothing if no sync clear/set on this register
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the cell in -cell.

-direction *moveDirection*

Specifies the direction of the register move across the given cell (forward or backward).

-class ***className***

Specifies the retiming class to which the moving registers belong.

-cell ***cellName***

Specifies the name of the cell instance over which registers move.

-input ***Pin:regName:AsyncState:SyncState***

For forward direction, this specifies the existing register regName that drives the input pin Pin of the cell cellName. For backward direction, this specifies the new after move register with the name regName that will drive the input pin Pin of the cell cellName. AsyncState (number between 0 and 7) specifies the async reset state (connections to its AC, AS pins) of this particular register relative to Class leader. SyncState (number between 0 and 7) specifies the sync reset state (connections to its SC, SS pins) of this particular register relative to Class leader. Multiple occurrences of this option are legal.

-output ***Pin:regName:AsyncState:SyncState***

For forward direction, this specifies the new after move register with the name regName that will be driven by the output pin Pin of the cell cellName. For backward direction, this specifies the existing register regName that is driven by the output pin Pin of the cell cellName. AsyncState (number between 0 and 7) specifies the async reset state (connections to its AC, AS pins) of this particular register relative to Class leader. SyncState (number between 0 and 7) specifies the sync reset state (connections to its SC, SS pins) of this particular register relative to Class leader. Multiple occurrences of this option are legal.

DESCRIPTION

This guide mode command describes how registers move over a given logic cell instance and the reset state of these registers before and after the move. For example, if one register is moving over an inverter, then its clear connection has to move to set. Otherwise, this move introduces a difference.

EXAMPLES

The following example shows a forward move across NAND gate instance *C1* in design named *test* of registers *Reg1* and *Reg2* with class *Class_1* (lets assume this class has async clear/set and no sync clear/set).

```
prompt> guide_retiming_move \
-design {test} \
-direction {svfForward}
-class {Class_1} \
-cell {C1} \
-input {A:Reg1:2:} \
-input {A:Reg2:2:} \
-output {Z:NewReg1:4:}
```

SEE ALSO

```
guide_retiming_pinmap(2)
guide_retiming_unmap(2)
guide_retiming_seqmap(2)
guide_retiming_class(2)
guide_retiming_cross(2)
```

guide_retiming_multibit

Indicates if a multibit register is split into one-bit registers.

SYNTAX

```
guide_retiming_multibit
   -design { designName }
   -multiName { previousName }
   -splitNames { newNames }
```

Data Types

```
   designName string
   previousName string
   newNames string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the multibit register.

-multiName *previousName*

Specifies the name of the multibit register before being split.

-splitNames *changedName*

Specifies the name of the registers after being split.

DESCRIPTION

Use this guide mode command to indicate that a multibit register is split into one-bit registers.

EXAMPLES

```
fm_shell (guide)> guide_retiming_multibit \
   -design { test } \
   -multiName { example_reg[0:3] } \
   -splitNames { Q3:example_reg[0:3]_Q3 } \
   -splitNames { Q2:example_reg[0:3]_Q2 } \
   -splitNames { Q1:example_reg[0:3]_Q1 } \
   -splitNames { Q0:example_reg[0:3]_Q0 }
1

fm_shell (guide)> guide_retiming_multibit \
   -design { test } \
   -multiName { my_reg[0:2] } \
   -splitNames { Q2:my_reg2 Q1:my_reg1 Q0:my_reg0 }
1
```

SEE ALSO

```
svf_retiming(3)
guide_retiming(2)
guide_retiming_decompose(2)
guide_retiming_finished(2)
```

guide_retiming_pinmap

Describes a mapping between pins of a techcell register and its internal SEQ pins.

SYNTAX

```
guide_retiming_pinmap
   -design {designName}
   -cell {cellName}
   -pinmap {Bit TechCellPin SEQpin Inversion}
   or
   -pinmap {Bit TechCellPin Constant NA}
```

Data Types

```
   designName string
   cellName string
   Bit integer
   TechCellPin string
   SEQpin string
   Inversion either svfTrue or svfFalse
   Constant either svfZero or svfOne for inputs, svfNC for outputs meaning No Connection
   NA string which is ignored
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the cell in -cell.

-cell *cellName*

Specifies the name of the cell instance of the techcell. We are only interested in the techcell not the instance. The linked design of the instance is our techcell.

-pinmap *Bit TechCellPin SEQpin Inversion*

Specifies, for the bitnumber Bit for multibit, how is the techcell pin named TechCellPin connects to the internal SEQ pin. If the connection has inversion, then svfTrue is given for the Inversion. One important thing to note here is that if both AC and AS is used on this register, then one needs to specify the contention (AC=AS=1) state. In such a case, instead of just AC as SEQpin we use ACL meaning L (low, 0) state for the contention, and we use ACH meaning H (high, 1) state for the contention. Also, AS needs to be given as ASL or ASH. Samething applies to sync SC SS contention (SCL/SCH for SC, and SSL/SSH for SS). If the TechCellPin is supposed to connect to a constant, then the second form of -pinmap is used. Multiple occurrences of this option are legal.

-pinmap *Bit TechCellPin Constant NA*

Specifies if the techcell register pin TechCellPin is a constant (svfZero or svfOne) for inputs, or No Connection (svfNC) for outputs. Multiple occurrences of this option are legal.

DESCRIPTION

This guide mode command describes the semantics of each techcell register pin in terms of its connection or "equivalent" connection to its internal SEQ pin. For multibit techcell register internal SEQs are indicated by their bit number. For single bit registers Bit is given as 0.

EXAMPLES

The following example shows the command used with a design named *test* on a techcell instance *reg1*.

```
prompt> guide_retiming_pinmap \
-design {test} \
-cell {reg1} \
-pinmap {0 DT SD svfFalse} \
-pinmap {0 Q Q svfFalse} \
-pinmap {0 RN AC svfTrue} \
-pinmap {0 SET AS svfFalse} \
-pinmap {0 EN SL svfFalse} \
-pinmap {0 CK CLK svfFalse} \
-pinmap {0 SE svfZero NA} \
-pinmap {0 SI svfZero NA}
```

SEE ALSO

```
guide_retiming_unmap(2)
```

```
    guide_retiming_seqmap(2)
    guide_retiming_class(2)
    guide_retiming_cross(2)
    guide_retiming_move(2)
```

guide_retiming_seqmap

Describes a mapping between pins of external logic around the SEQ and the function of an unused pin of the SEQ, such that the extra logic can be removed and the pin is used instead. For example, Sync Clear is iplemented via an external logic to pull the data line to 0 when clear is active, and we want to remove the extra logic and use SC pin instead.

SYNTAX

```
guide_retiming_seqmap
   -design {designName}
   -cell {cellName}
   -seqmap {Cell:Pin SEQpin Inversion}
   or
   -seqmap {Cell:Pin Constant NA}
```

Data Types

```
designName string
cellName string
Cell string
Pin string
SEQpin string
Inversion either svfTrue or svfFalse
Constant either svfZero or svfOne
NA string which is ignored
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the cell in -cell.

-cell *cellName*

Specifies the name of the SEQ that is created by guide_retiming_unmap.

-seqmap *Cell:Pin SEQpin Inversion*

Specifies that the pin given by Cell:Pin pair implements the SEQpin functionality. It might have inverted relation indicated by svfTrue on the Inversion entry. One important thing to note here is that if both AC and AS is used on this register, then one needs to specify the contention (AC=AS=1) state. In such a case, instead of just AC as SEQpin we use ACL meaning L (low, 0) state for the contention, and we use ACH meaning H (high, 1) state for the contention. Also, AS needs to be given as ASL or ASH. Samething applies to sync SC SS contention (SCL/SCH for SC, and SSL/SSH for SS). If the Cell:Pin is supposed to connect to a constant, then the second form of -seqmap is used. Multiple occurrences of this option are legal.

-seqmap *Cell:Pin Constant NA*

Specifies if the pin Cell:Pin is driven by a constant (svfZero or svfOne). Multiple occurrences of this option are legal.

DESCRIPTION

This guide mode command describes the pin and function correspondence of extra logic around a SEQ and unused pins of that SEQ. Such that we can remove the extra logic and utilize the unused SEQ pins instead.

EXAMPLES

The following example shows the command used with a design named *test* and SEQ *reg1*, in order to pull-in the reset logic.

```
prompt> guide_retiming_seqmap \
-design {test} \
-cell {reg1} \
-seqmap {U1:S SC svfFalse} \
-seqmap {U1:A svfZero NA} \
-seqmap {U1:B SD svfFalse}
```

SEE ALSO

```
guide_retiming_pinmap(2)
guide_retiming_unmap(2)
guide_retiming_class(2)
guide_retiming_cross(2)
```

```
guide_retiming_move(2)
```

guide_retiming_unmap

Describes how we actually apply the pinmap description on the given register instance.

SYNTAX

```
guide_retiming_unmap
   -design {designName}
   -cell {cellName}
   -namemap {Bit RegName}
```

Data Types

```
designName string
cellName string
Bit integer
RegName string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design that contains the cell in -cell.

-cell *cellName*

Specifies the name of the cell instance of the techcell.

-namemap *Bit RegName*

Specifies the name of the SEQ that comes out of the pinmap conversion of this instance -cell. For multibit registers the Bit number is given. For single-bit registers Bit is given as 0. Multiple occurrences of this option are legal.

DESCRIPTION

This guide mode command describes the names of the SEQs created by application of guide_retiming_pinmap description. For multibit registers the Bit number is given. For single-bit registers Bit is given as 0.

EXAMPLES

The following example shows the command used with a design named *test* and on register instance "reg1" which is a multibit (4 bit) register.

```
prompt> guide_retiming_unmap \
-design {test} \
-cell {reg1} \
-namemap {0 REG_0} \
-namemap {1 REG_1} \
-namemap {2 REG_2} \
-namemap {3 REG_3}
```

SEE ALSO

```
guide_retiming_pinmap(2)
guide_retiming_seqmap(2)
guide_retiming_class(2)
guide_retiming_cross(2)
guide_retiming_move(2)
```

guide_rewire

Identifies registers or pins that are merged.

SYNTAX

```
status guide_rewire
   [ -design designName ]
     -from fromList
     -to toReg
```

Data Types

```
designName string
fromList string
toReg string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the merged registers.

-from *fromList*

Specifies a list of the merged registers or pins.

-to *toReg*

Specifies the final register or pin.

DESCRIPTION

Use the **guide_rewire** command to identify registers or pins that are merged. This command is similar to the **guide_reg_merging** command, but has the additional functionality of allowing registers at different levels of the hierarchy to be merged. Only pins that are directly driven by registers are allowed to be merged. This command is a superset of the **guide_reg_merging** command.

EXAMPLES

```
fm_shell (guide)> guide_rewire \
                  -design top \
                  -from { Q1 Q3 } \
                  -to Q2
1
```

SEE ALSO

```
guide(2)
guide_reg_merging(2)
guide_reg_eqop(2)
```

guide_scan_input

Identifies a port or black box pin that should be tied to a constant to disable scan.

SYNTAX

```
guide_scan_input
   -design design_name
   -disable_value 0 | 1
   -ports { port_name ... }
   -pins { pin_name ... }
```

Data Types

```
design_name string
port_name string
pin_name string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design

Specifies the name of the design whose ports or pins should be tied to a constant.

-disable_value

Specifies the constant value (0 or 1) that should be applied to the input port or BBOX output pin in order to disable scan circuitry that is inserted.

-ports

Specifies a Tcl list of input port names that should be tied to the value specified by using the -*disable_value* option.

-pins

Specifies a Tcl list of BBPX pin names that should be tied to the value specified by using the -*disable_value* option..

DESCRIPTION

Use this guide mode command to specify that some inputs should be held constant in order to disable scan circuitry. This is used when scan circuitry is inserted or scan chains are reordered. This command is ignored if either the **synopsys_auto_setup** or **svf_scan** variables are not set to *true*.

EXAMPLES

To set the port test_se of design top, to 0:

```
fm_shell (guide)> guide_scan_input -design top -disable_value 0 -ports { test_se }
```

SEE ALSO

```
guide(2)
guide_scan_output(2)
synopsys_auto_setup(3)
```

guide_set_rounding

Specifies the initial rounding information for multipliers.

SYNTAX

```
guide_set_rounding
   -design designName
   -cells cellName
   -rounding { ExtPos [ IntPos ] }
```

Data Types

```
   designName string
   cellName string
   ExtPos int
   intPos int
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the multiplier.

-cells *cellName*

Specifies the name of the multiplier cell.

-rounding { *ExtPos IntPos* }

Specifies the external and internal rounding positions.

DESCRIPTION

Use this guide mode command to specify the internal and external rounding modifications applied to a multiplier.

EXAMPLES

```
fm_shell (guide)> guide_set_rounding \
                  -design { top } \
                  -cells { mult_123 } \
                  -rounding { 12 4 }
1
```

SEE ALSO

```
guide(2)
set_dp_int_round(2)
```

guide_share

Describes high-level resource sharing optimizations.

SYNTAX

```
guide_share
   { -origin originName }
   { -design designName }
   { -input inputList }
   { -output outputList }
   { -control controlList }
   { -pre_resource preResourceList }
   { -pre_assign preAssignList }
   { -post_resource postResourceList }
   { -post_assign postAssignList }
```

Data Types

```
originName string
designName string
inputList string
outputList string
controlList string
preResourceList string
preAssignList string
postResourceList string
postAssignList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-origin *originName*

Specifies the name indicating where the optimization came from.

-design *designName*

Specifies the name of the design in which the optimization takes place.

-input *inputList*

Lists graph inputs.

-output *outputList*

Lists graph outputs.

-control *controlList*

Lists control inputs.

-pre_resource *preResourceList*

Lists resources in the optimization pre-graph.

-pre_assign *preAssignList*

Lists assignments in the optimization pre-graph.

-post_resource *postResourceList*

Lists resources in the optimization post graph.

-post_assign *postAssignList*

Lists assignments in the optimization post graph.

DESCRIPTION

Use this guide mode command to specify high-level resource sharing optimizations.

EXAMPLES

```
fm_shell (guide)> guide_share \
  -origin { ExTra_mutex } \
  -design { dp } \
  -input { 8 I1 } \
  -input { 8 I2 } \
  -output { 8 O1 } \
  -output { 8 O2 } \
  -control { C1 = add_10 } \
  -control { C2 = sub_10 } \
  -pre_resource { { 8 } add_10 = ADD { { I1 ZERO 8 } { I2 ZERO 8 } } } \
  -pre_resource { { 8 } sub_10 = SUB { { I1 ZERO 8 } { I2 ZERO 8 } } } \
  -pre_assign { O1 = { add_10 ZERO 8 } } \
  -pre_assign { O2 = { sub_10 ZERO 8 } } \
  -post_resource { { 8 } addsub_10_addsub = ADDSUB { { addsub_10_sel_1 ZERO 8 }
     { addsub_10_sel_2 ZERO 8 } { addsub_10_sel_3 } } } \
  -post_resource { { 8 } addsub_10_sel_1 = SELECT { { C1 } { C2 } { I1 ZERO 8 }
     { I1 ZERO 8 } } } \
  -post_resource { { 8 } addsub_10_sel_2 = SELECT { { C1 } { C2 } { I2 ZERO 8 }
     { I2 ZERO 8 } } } \
  -post_resource { { 1 } addsub_10_sel_3 = SELECT { { C1 } { C2 } { `b0 }
     { `b1 } } } \
  -post_assign { O1 = { addsub_10_addsub ZERO 8 } } \
  -post_assign { O2 = { addsub_10_addsub ZERO 8 } }
1
```

SEE ALSO

```
guide(2)
```

guide_transformation

Describes design transformations.

SYNTAX

```
guide_transformation
   -design designName
   -type type
   -input inputList
   -output outputList
   [ -control controlList ]
   [ -virtual virtualList ]
   [ -pre_resource preResourceList ]
   [ -pre_assign preAssignList ]
   [ -post_resource postResourceList ]
   [ -post_assign postAssignList ]
   [ -datapath datapathList ]
```

Data Types

```
designName string
type string
inputList string
outputList string
controlList string
virtualList string
preResourceList string
preAssignList string
postResourceList string
postAssignList string
datapathList string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design ***designName***

Specifies the name of the design to be transformed.

-type ***type***

Specifies the type of transformation: share, tree, map, or merge.

-input ***inputList***

Lists transformation inputs.

-output ***outputList***

Lists transformation outputs.

-control ***controlList***

Lists transformation control signals.

-virtual ***virtualList***

Lists transformation virtual signals.

-pre_resource ***preResourceList***

Lists resources in the transformation pre-graph.

-pre_assign ***preAssignList***

Lists assignments in the transformation pre-graph.

-post_resource ***postResourceList***

Lists resources in the transformation post graph.

-post_assign ***postAssignList***

Lists assignments in the transformation post graph.

-datapath ***datapathList***

Lists datapath elements in the transformation.

DESCRIPTION

Use this guide mode command to describe high-level design transformations.

EXAMPLES

```
fm_shell (guide)> guide_transformation \
                      -design general_tree \
                      -type tree \
                   -input { 4 src1 4 src2 4 src4 } \
                   -output { 6 O1 } \
                   -control { ctrl1 = cond ctrl2 } \
                   -pre_resource { { 5 5 } add_7 = DIV { { src1 2 4 }
          { src2 2 3 ZERO 3 } } } \
                   -pre_resource { { 5 5 } add_8 = DIV { { src1 2 4 }
          { src2 2 4 ZERO 5 } } } \
                   -pre_resource { { 5 5 } add_9 = DIV { { src1 ZERO 5 }
          { src2 ZERO 5 } } } \
                   -pre_resource { { 6 4 4 } sub_9 = USUB { { add_9 ZERO 6 }
          { src4 ZERO 6 } } } \
                   -pre_assign { O1 = { sub_9 } } \
                   -post_resource { { 6 7 4 3 } sub_1_root_sub_9 = USUB {
          { src1 ZERO 6 } { src4 ZERO 6 } } } \
                   -post_resource { { 6 7 3 2 5 } add_0_root_sub_9 = UADD {
          { src2 ZERO 6 } { sub_1_root_sub_9 } } } \
                   -post_assign { O1 = { add_0_root_sub_9 } }
1
```

SEE ALSO

guide(2)

guide_ungroup

Removes hierarchy from design objects.

SYNTAX

```
guide_ungroup
     -design designName
   [ -instance instanceName ]
   [ -cells cellList ]
   [ -all ]
   [ -small ]
   [ -flatten ]
   [ -start_level ]
   [ ungroupBlock ]
```

Data Types

```
designName string
instanceName string
cellList string
ungroupBlock string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the objects.

-instance *instanceName*

Specifies the name of the instance containing the objects.

-cells *cellList*

Lists the cells being ungrouped.

-all, -small, -flatten, -start_level

Specify the Design Compiler ungroup options.

ungroupBlock

Specifies a space separated list of old-cell new-cell pairs.

DESCRIPTION

Use this guide mode command to remove hierarchy from design objects.

EXAMPLES

```
fm_shell (guide)> guide_ungroup \
              -design shift8 \
                 -instance shift16/shift8_i_1 \
                 -cells { foo } \
                 -flatten \
                 { { shift4_i_1/shift2_i_1/dout_reg dout_reg } \
                 { shift4_i_1/shift2_i_1/state1_reg state1_reg } \
                 { shift4_i_1/shift2_i_2/dout_reg dout_reg1 } \
                 { shift4_i_1/shift2_i_2/state1_reg state1_reg1 } \
                 { shift4_i_2/shift2_i_2/state1_reg state1_reg3 } }
1
```

SEE ALSO

```
guide(2)
```

guide_uniquify

Creates unique instances for design objects.

SYNTAX

```
guide_uniquify
   -design designName
   [ uniquifyBlock ]
```

Data Types

```
designName string
uniquifyBlock string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the objects being uniquified.

uniquifyBlock

Specifies a space separated list of old-cell new-design pairs.

DESCRIPTION

Use this guide mode command to create unique instances for design objects.

EXAMPLES

```
fm_shell (guide)> guide_uniquify -design shift4 \
                  { { shift2_i_1 shift2_0 } { shift2_i_2 shift2_1 } }
1
```

SEE ALSO

```
guide(2)
```

guide_ununiquify

Folds uniquified instances back to their original designs.

SYNTAX

```
guide_ununiquify
   -design designName
   [ ununiquifyBlock ]
```

Data Types

```
designName string
ununiquifyBlock string
```

ENABLED SHELL MODES

Guide

ARGUMENTS

-design *designName*

Specifies the name of the design containing the objects being folded.

ununiquifyBlock

Specifies a space separated list of old-cell new-design pairs.

DESCRIPTION

Use this guide mode command to fold uniquified instances back to their original designs.

EXAMPLES

```
fm_shell (guide)> guide_ununiquify \
                  -design top \
                  { { U2 mid } \
                  { U1 mid } \
                  { U2/U3 bot } \
                  { U1/U3 bot } }
1
```

SEE ALSO

```
guide(2)
```

help

Returns information about Formality shell commands.

SYNTAX

```
help
   [ -verbose ]
   [ -groups ]
   [ pattern ]
```

Data Types

```
pattern string
```

ARGUMENTS

-verbose

Displays all the **help** command options and detailed messages.

-groups

Displays command groups only.

pattern

Specifies a Formality shell *command*.

DESCRIPTION

Use this command to see the help information for a Formality shell command.

This command does not return syntax information about fm_shell commands. To display syntax information about a command, use:

```
command -help
```

To display information about a particular command, specify the name of the *command* as the argument. When you don't specify a command name, Formality displays an alphabetical list of all available Formality shell commands.

EXAMPLES

This example displays the help information for the **read_db** command:

```
fm_shell> help read_db
read_db -- Read one or more db files
```

This example displays syntax and help information for the **read_db** command:

```
fm_shell> read_db -help
Usage: read_db    # Read one or more db files
[-container container_name]    (Read into named container)
[-libname libname]     (Library or design library name)
file_names             (List of files to read)
```

SEE ALSO

```
man(2)
```

help_attributes

Display help for attributes and object types.

SYNTAX

```
string help_attributes [-verbose]
   [class_name]
   [attr_pattern]
```

Data Types

```
string  class_name
string  attr_pattern
```

ARGUMENTS

-verbose

Display attribute properties.

class_name

Object class to retrieve help for. Valid classes are *lib, lib_cell, lib_pin, design, port, cell, pin, and net.*

attr_pattern

Show attributes matching pattern.

DESCRIPTION

The **help_attributes** command is used to get quick help for the set of available attributes. When this command is run with no arguments a brief informational message is printed followed by the available object classes.

EXAMPLES

The following example lists net attributes ending with "name".

```
prompt> help_attributes net *name
Available net attributes:
 container_name         # string
 full_name              # string
 library_name           # string
 name                   # string
 parent_name            # string
 path_name              # string
```

SEE ALSO

```
get_attribute(2)
list_attributes(2)
```

history

Displays or modifies the commands recorded in the history list.

SYNTAX

```
history
   [-h]
   [-r]
   [args]
```

Data Types

```
args string
```

ARGUMENTS

-h

Displays the history list without the leading numbers. Use this switch to create scripts from the existing history. You can then source the script by using the **source** command. You can use this switch only with a single numeric argument. Note that this switch is not a standard extension to Tcl.

-r

Reverses the order of the output. The most recent entries are displayed first rather than the oldest entries first. You can use this switch only with a single numeric argument. Note that this switch is not a standard extension to Tcl.

args

Specifies additional arguments to the **history** command.

DESCRIPTION

Use this command to perform several operations related to the recently executed commands that are recorded in the history list.

Each of the recorded commands is referred to as an *event*. The most commonly used forms of the command are illustrated in the following section. You can combine each with either the **-h** or *-r* switches, but not both. When you don't use any argument, the **history** command returns a formatted string, intended for you to read, giving the event number and contents for each of the events in the history list.

If a single, integer argument count is specified, only the most recent count events are returned. Note that this option is not a standard extension to Tcl.

By default, 20 events are retained in the history list. You can change the length of the history list by using the following command:

```
fm_shell> history keep integer
```

Tcl supports several additional forms of the **history** command. See the History Revision section.

EXAMPLES

The following examples illustrate the basic use of the **history** command.

To limit the number of events by using a single numeric argument:

```
fm_shell> history 3
   7 set base_name "my_file"
   8 set fname [format "%s.db" $base_name]
   9 history 3
```

To view the history list in reverse order:

```
fm_shell> history -r 3
   9 history -r 3
   8 set fname [format "%s.db" $base_name]
   7 set base_name "my_file"
```

To remove the leading numbers from each line in the history:

```
fm_shell> history -h 3
   set base_name "my_file"
   set fname [format "%s.db" $base_name]
   history -h 3
```

Advanced Tcl History

The **history** command has many advanced features. The recently executed commands are recorded in a history list. Each of these recorded commands are referred to as an event. You can specify an event by using any of the following forms:

- A number: A positive integer refers to the event associated with the number. All events are numbered starting at 1. A negative integer selects an event relative to the current event. For example, specifiying *-1* refers to the previous event and *-2* to the one before that.
- A string: Selects the most recent event that matches the string. An event matches the string either if the string is the same as the first characters of the event or if the string matches the

event in the sense of the **string match** command.

The **history** command can take any of the following forms:

- **history**
  Same as "**history** info", described later in this man page.
- **history** add <command> [exec]
  Adds the "command" argument to the **history** list as a new event. If you specify (or abbreviate) **exec**, the command also executes and its result is returned. If you do not specify **exec**, the command returns an empty string.
- **history** change <newValue> [event]
  Replaces the value recorded for an event with the *newValue*. The *event* argument specifies the event to replace and defaults to the current event (not event -1). This intention of the command is for your use with commands that implement new forms of history substitution to replace the current event (which invokes the substitution) with the command created through substitution. The return value is an empty string.
- **history** event <event>
  Returns the value of the event specified by using the *event* argument. The defualt value of the *event* argument is -1. This command revises the history. See the Advanced Tcl History section for details.
- **history** info [count]
  Returns a formatted string (intended for you to read) giving the event number and contents for each of the events in the history list except the current event. If you specify *count*, only the most recent *count* number if events are returned.
- **history** keep <count>
  You can use this command to change the size of the history list to a number of events specified by using the *countfP argument. By defualt, 20 events are retained in the history list. This command returns an empty string.*
- **history** nextid
  Returns the number of the next event to record in the **history** list. Use this for things like printing the event number in command-line prompts.
- **history** redo [event]
  Re-executes the command indicated by the *event* argument and returns its result. By default the *event* argument is set to -1. This command results in history revision. See History Revision for details.
- **history** substitute <old> <new> <event>
  Retrieves the command specified by using the *event* argument. By default the *event* argument is set to -1 and replaces any occurrences of "old" or "new" in the command. Only simple character equality is supported, wild cards are not supported. This use also executes the resulting command, and returns the result. This command results in **history** revision. See History Revision for details.
- **history** words <selector> <event>
  Retrieves history from the command specified by the *event* argument and the words specified by using the *selector* argument. This command returns the words in a string separated by spaces. The *selector* argument has three forms. If it is a single number, it selects the word given by that number (0 for the command name, 1 for its first argument, and so on). If it consists of two

numbers separated by a dash, it selects all the arguments between the two apecified numbers. Otherwise the *selector* is treated as a pattern; all words matching that pattern (in the sense of "string match") are returned. In the numeric forms you can use $ to select the last word of a command.

For example, suppose the most recent command in the **history** list is

```
format  {%s is %d years old} Alice [expr $ageInMonths/12]
```

The following are some **history** commands and the results they would produce.

```
history words $      [expr $ageInMonths/12]
history words 1-2    {%s is %d years old} Alice
history words *a*o*  {%s is %d years old} [expr $ageInMonths/12]
```

History words results in **history** revision.

- **History Revision:**
  The **history** options "event", "redo, "substitute, and "words" result in "**history** revision". When you invoke one of these options, the current event is modified to eliminate the **history** command and replaced with the result of the **history** command. For example, suppose that the most recent command in the **history** list is:

  ```
  set a [expr $b+2]
  ```

and that the next command invoked is one from the left side of the following table. The command actually recorded in the **history** event is the corresponding one on the right side of the table.

```
history redo         set a [expr $b+2]
history s a b        set b [expr $b+2]
set c [history w 2]  set c [expr $b+2]
```

History revision is needed because event specifiers like -1 are only valid at a particular time; after more events are added to the **history** list, a different event specifier is needed.

History revision occurs even when you invoke **history** indirectly from the current event (such as, you type a command that invokes a Tcl procedure that invokes history): the top-level command whose execution eventually results in a **history** command is replaced.

To invoke commands like "**history** words" without **history** revision, you can use "**history** event" to save the current **history** event then use "**history** change" to restore it later.

SEE ALSO

index_collection

Given a collection and an index into it, if the index is in range, create a new collection containing only the single object at the index in the base collection. The base collection remains unchanged.

SYNTAX

```
collection index_collection
   collection1
   index
```

Data Types

```
collection1         collection
index               int
```

ARGUMENTS

collection1

Specifies the collection to be searched.

index

Specifies the index into 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 containing only that object. The index operation is done in constant time - it is independent of the number of elements in the collection, or the specific index.

The range of indices is 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.

You can use the empty string for the *collection1* argument. However, by definition, any index into the empty collection is invalid. Therefore, using the **index_collection** command with the empty collection always generates the empty collection as a result and generates an error message.

Note that not all collections can be indexed.

EXAMPLES

The following example uses the **index_collection** command to extract the first object of a collection.

```
prompt> set c1 [get_cells {u1 u2}]
{r:/WORK/top/u1 r:/WORK/top/u2}
prompt> query_objects [index_collection $c1 0]
{r:/WORK/top/u1}
```

SEE ALSO

```
collections(2)
query_objects(2)
sizeof_collection(2)
```

insert_inversion

Creates an inversion at the specified net, pin, or port.

SYNTAX

```
status insert_inversion
   [-type ID_type]
   objectID
```

Data Types

```
ID_type string
objectID string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

-type *ID_type*

Specifies the object type. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following object types:

- *pin*
- *port*
- *net*

objectID

Specifies the design object where you want to insert inversion. You can specify a primary port, pin, or net unless they are bidirectional. If the name you specify resolves to multiple objects with identical names, and if you do not specify the **-type** option, the command searches for the specified object in the following order: pin, port, and net.

DESCRIPTION

This command inserts inversion at the specific object. You must specify the object as a folded design path. Inversion can be inserted on ports, pins, and nets. Inversion cannot be inserted on bidirectional ports or pins. If there is a bidirectional pin connected to a specified net, the command reports an error and exits. Note that if a specified net has multiple fanout, they are all inverted. Specify a primary port or pin if you do not want to invert all the fanouts or fanins of a net.

The **Insert_inversion** command is available from the GUI in the "Setup" menu of the logic cone schematics. You must select a net, pin, or port before clicking "Insert inversion" on the "Setup menu". Note that you can select the diamond shaped hierarchy separator to make a pin selection for the **insert_inversion** command. The GUI queues the command, because the **insert_inversion** command must be executed in the setup mode.

NOTE that the **insert_inversion** command is made on the actual design, so the inversion is inserted in all instantiations of the design. A warning message is reported if the inversion is inserted in more places than the selected net, pin, or port. No warning will be given from the command line. The GUI default for net selection crosses the hierarchy separators making it possible to select multiple nets segments. This can be turned off by editing the GUI preferences, and setting "Skip Hierarchical Crossings" to "false". If multiple net segments are selected across the hierarchy, then the driving net is chosen by the GUI for the **insert_inversion** command.

Use the **report_inversion** command to view a list of inversion objects.

EXAMPLES

The following example shows how to insert inversion on a primary output port out1.

```
fm_shell (setup)> insert_inversion i:/WORK/top/out1 -type port
Inversion created at 'i:/WORK/top/out1_FM_INV'
1
```

The following example inserts inversion on an input pin i1.

```
fm_shell (setup)> insert_inversion i:/WORK/top/bot1/i1 -type pin
Inversion created at 'i:/WORK/top/i1_FM_INV'
```

```
    1
```

The following example inserts inversion on a net in2.

```
    fm_shell (setup)> insert_inversion i:/WORK/top/in2 -type net
    Inversion created at 'i:/WORK/top/in2_FM_INV'
    1
```

SEE ALSO

```
remove_inversion(2)
report_inversion(2)
```

invert_pin

Inverts one or more pins on a primitive cell.

SYNTAX

```
status invert_pin
   pin_list
```

Data Types

```
pin_list list
```

RETURN VALUE

The **invert_pin** command returns a status of 1 if it succeed and 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

pin_list

Inverts the specified pins. You can specify either object IDs or instance-based paths.

Inversion of a pin is visible to all instances of that pin. To affect a specific instance-based path, uniquify that instance of the design before you run this command.

DESCRIPTION

This command inverts the specified pins on primitive cells. Only pins on non-SEQ primitive cells are inverted.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

This command is specific to Formality. It is not supported by other Synopsys tools. Formality will warn about this the first time the command is used.

EXAMPLES

The following example shows how to create a new AND cell in design r:/WORK/mid and inverts the output pin effectively making it a NAND gate.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_primitive my_gate AND
fm_shell (setup)> invert_pin my_gate/OUT
```

This example shows how to create a new XOR cell in design i:/WORK/top/m1/b1 and inverts the output pin effectively making it a XNOR gate. If the instance-based path i:/WORK/top/m1/b1 is not unique, all instances are affected.

```
fm_shell (setup)> current_instance r:/WORK/top/m1/b1
fm_shell (setup)> create_primitive my_xnor XOR
fm_shell (setup)> invert_pin my_xnor/OUT
```

SEE ALSO

```
create_primitive(2)
current_design(2)
current_instance(2)
edit_design(2)
```

is_false

Tests the value of a specified variable, and returns 1 if the value is 0 or the case-insensitive string **false**; returns 0 if the value is 1 or the case-insensitive string **true**.

SYNTAX

```
status is_false
   value
```

Data Types

```
value       string
```

ARGUMENTS

value

Specifies the name of the variable whose value is to be tested.

DESCRIPTION

This command tests the value of a specified variable, and returns 1 if the value is either 0 or the case-insensitive string **false**. The command returns 0 if the value is either 1 or the case-insensitive string **true**. Any value other than 1, 0, **true**, or **false** generates a Tcl error message.

The **is_false** command is used in writing scripts that test Boolean variables that can be set to 1, 0, or the case-insensitive strings **true** or **false**. When such variables are set to **true** or **false**, they cannot be tested in the negative in an **if** statement by simple variable substitution, because they do not constitute a true or false condition. The following example is not legal Tcl:

```
set x FALSE
if { !$x } {
  set y TRUE
}
```

This results in a Tcl error message, indicating that you cannot use a non-numeric string as the operand of "!". So, although you can test the positive condition, **is_false** allows you to test both conditions safely.

EXAMPLES

The following example shows the use of the **is_false** command:

```
prompt> set x TRUE
TRUE

prompt> if { ![is_false $x] } {
?         set y TRUE
?       }
TRUE

prompt>
```

SEE ALSO

```
expr(2)
if(2)
is_true(2)
```

is_true

Tests the value of a specified variable, and returns 1 if the value is 1 or the case-insensitive string **true**; returns 0 if the value is 0 or the case-insensitive string **false**.

SYNTAX

```
status is_true
   value
```

Data Types

```
   value      string
```

ARGUMENTS

value

Specifies the name of the variable whose value is to be tested.

DESCRIPTION

This command tests the value of a specified variable, and returns 1 if the value is either 1 or the case-insensitive string **true**. The command returns 0 if the value is either 0 or the case-insensitive string **false**. Any value other than 1, 0, **true**, or **false** generates a Tcl error message.

The **is_true** command is used in writing scripts that test Boolean variables that can be set to 1, 0, or the case-insensitive strings **true** or **false**. When such variables are set to **true** or **false**, they cannot be tested in the negative in an **if** statement by simple variable substitution, because they do not constitute a true or false condition. The following example is not legal Tcl:

```
set x TRUE
if { !$x } {
  set y FALSE
}
```

This results in a Tcl error message, indicating that you cannot use a non-numeric string as the operand of "!". So, although you can test the positive condition, **is_true** allows you to test both conditions safely.

EXAMPLES

The following example shows the use of the **is_true** command:

```
prompt> set x FALSE
FALSE

prompt> if { ![is_true $x] } {
?         set y FALSE
?       }

FALSE

prompt>
```

SEE ALSO

```
expr(2)
if(2)
is_false(2)
```

library_verification

Starts the library verification mode.

SYNTAX

```
library_verification
   [ mode ]
```

Data Types

```
mode string
```

ARGUMENTS

mode

Specifies the library verification mode. Specify one of the following values for the *mode* argument:

- VERILOG_VERILOG - verifies two versions of a Verilog simulation library
- VERILOG_DB - verifies a Verilog simulation library against a library in .db file format
- VERILOG_PWRDB - verifies Verilog cells in the reference library against the power version of the cells in the implementation library that is the in .db file format
- DB_DB - verifies two versions of a .db file format library
- DB_VERILOG - verifies a .db format library against a Verilog simulation library
- PWRDB_VERILOG - verifies the power version of cells in the reference library in .db file format against the Verilog cells in the implementation library
- NONE - exits the library verification mode

DESCRIPTION

This command starts the library verification mode to read and verify library cells.

To restart the fm_shell design environment, use the *NONE* mode. When the *NONE* mode is specified, the tool removes all containers containing the library cells and exits the library verification mode.

The following commands are available in the library verification mode:

- **read_verilog** - reads Verilog library cells
- **read_db** - reads the .db file format libraries
- **report_cell_list** - reports cells
- **select_cell_list** - selects cells
- **verify** - verifies all library cells
- **report_status** - reports the status of verification
- **debug_library_cell** - debugs failing cell in library verification mode
- **write_library_debug_scripts** - debugs failing cells in the formal verification mode

For more information on these commands, see the man pages of each command.

EXAMPLES

SEE ALSO

```
debug_library_cell(2)
read_db(2)
read_verilog(2)
report_status(2)
select_cell_list(2)
verify(2)
write_library_debug_scripts(2)
```

license_users

Lists the current users of the Synopsys licensed features.

SYNTAX

```
status license_users
   [feature_list]
```

Data Types

```
feature_list list
```

ARGUMENTS

feature_list

Lists the licensed features for which to obtain the information. The *feature_list* argument might consist of a single value or a space-delimited list of values enclosed within braces ({}).

See the Synopsys Installation Guide: UNIX-Based Platforms for a list of features supported by the current release, or determine from your key file all of the features licensed at your site.

RETURN VALUE

The **license_users** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command displays information about all of the licenses, related users, and host names currently in use. If a feature list is specified, only information about those features is displayed.

The **license_users** command is valid only when Network Licensing is enabled.

EXAMPLES

In the following example, all users of Synopsys features are displayed:

```
fm_shell (setup)> license_users
carlson@node1                  Formality, Formality-Ultra
frodo@node3                    Design-Compiler, DC-Expert
```

SEE ALSO

```
check_license(2)
list_licenses(2)
get_license(2)
remove_license(2)
```

list_attributes

Lists currently defined attributes.

SYNTAX

```
string list_attributes
   [-application]
   [-class class_name]
   [-nosplit]
```

Data Types

```
   class_name       string
```

ARGUMENTS

-application

Lists application attributes as well as user-defined attributes (see DESCRIPTION).

-class *class_name*

Limit the listing to attributes of a single class. Valid classes are *lib, lib_cell, lib_pin, design, port, cell, pin, and net.*

-nosplit

Prevents line-splitting and facilitates writing software to extract information from the report output. Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

DESCRIPTION

The **list_attributes** command displays an alphabetically sorted list of currently defined attributes. There are two categories of attributes: application-defined and user-defined. Formality only has application-defined attributes; user-define attributes are not supported. By default, the **list_attributes** command lists user-defined attributes. This is useful for other tools, but for Formality the **-application** option must be used to obtain any useful output.

Using the **-application** option adds all application attributes to the listing. Note that there are many application attributes. It is often useful to limit the listing to a specific object class using the *class_name* option.

EXAMPLES

The following example lists all net attributes.

```
prompt> list_attributes -application -class net

****************************************
Report : List of Attribute Definitions
Version: ...
Date   : ...
****************************************

Properties:
    A - Application-defined
    S - Settable
    B - Subscripted

Attribute Name            Object     Type       Properties  Constraints
--------------------------------------------------------------------------------
container_name            net        string     A
full_name                 net        string     A
library_name              net        string     A
name                      net        string     A
object_class              net        string     A
parent_name               net        string     A
path_name                 net        string     A
type                      net        string     A
```

SEE ALSO

```
get_attribute(2)
help_attributes(2)
```

list_key_bindings

Lists the key bindings in the current editing mode

SYNTAX

```
list_key_bindings
   [ -nosplit ]
```

ARGUMENTS

-nosplit

Displays column text without splitting them.

DESCRIPTION

The **list_key_bindings** command displays the current key bindings in the current edit mode. To change the edit mode, use the **sh_line_editing_mode** variable in either the .synopsys_fm.setup file or in the fm_shell.

The text CTRL+K is read as `Control+K' and describes the character produced when the K key is pressed while the Control key is pressed.

The text META+K is read as `Meta+K' and describes the character produced when the k key is pressed while the Meta key is pressed. The Meta key is labeled ALT on many keyboards. On keyboards with two keys labeled ALT (usually to either side of the space bar), the ALT on the left side is generally set as the Meta key. The ALT key on the right might also be configured to work as a Meta key or configured as some other modifier, such as a Compose key for typing accented characters.

If you do not have a Meta or ALT key, or another key working as a Meta key, the identical keystroke can be generated by typing ESC first, and then typing k. Either process is known as metafying the k key.

Alternative key bindings work only in the VI alternate (command) mode.

SEE ALSO

```
sh_enable_line_editing(3)
sh_line_editing_mode(3)
```

list_libraries

Lists technology libraries that are currently loaded.

SYNTAX

```
list_libraries
```

DESCRIPTION

Use this command to list the technology libraries that are currently loaded. Shared libraries are listed first.

The **list_libraries** command returns one of the following:

- 0 to indicate failure
- List of library names

For more information about libraries and shared libraries, see the *Formality User Guide*.

EXAMPLES

This example assumes an environment that contains two containers, reference and implementation, and three libraries - *gtech*, *lca10k*, and *attlib*. In the example, *gtech* is a shared library because it is loaded into both reference and implementation containers. The other libraries, *lca10k* and *attlib*, are not shared libraries because they are loaded into individual containers.

```
fm_shell> list_libraries
gtech implementation:/lca500k ref:/lca500k
fm_shell>
```

SEE ALSO

```
remove_library(2)
report_containers(2)
report_libraries(2)
```

list_licenses

Displays a list of licenses currently checked out by the user.

SYNTAX

```
status list_licenses
   feature_list
```

Data Types

```
feature_list list
```

ARGUMENTS

The **list_licenses** command has no arguments.

RETURN VALUE

The **list_licenses** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command lists the licenses you currently have checked out. If more than one copy of a license key is checked out, the number of keys checked out is shown in parentheses following the license name.

EXAMPLES

The following example illustrates the use of **list_licenses**:

```
fm_shell (setup)> list_licenses
Licenses in use:
        Design-Compiler
        Formality
```

SEE ALSO

```
check_license(2)
license_users(2)
remove_license(2)
get_license(2)
```

lminus

Removes one or more named elements from a list and returns a new list.

SYNTAX

```
list lminus
   [-exact]
   original_list
   elements
```

Data Types

```
original_list     list
elements          list
```

ARGUMENTS

-exact

Specifies that the exact pattern is to be matched. By default, **lminus** uses the default match mode of **lsearch**, the **-glob** mode.

original_list

Specifies the list to copy and modify.

elements

Specifies a list of elements to remove from *original_list*.

DESCRIPTION

The **lminus** command removes elements from a list by using the element itself, rather than the index of the element in the list (as in **lreplace**). The **lminus** command uses the **lsearch** and **lreplace** commands to find the elements and replace them with nothing.

If none of the elements are found, a copy of *original_list* is returned.

The **lminus** command is often used in the translation of Design Compiler scripts that use the subtraction operator (-) to remove elements from a list.

EXAMPLES

The following example shows the use of the **lminus** command. Notice that no error message is issued if a specified element is not in the list.

```
prompt> set l1 {a b c}
a b c

prompt> set l2 [lminus $l1 {a b d}]
c

prompt> set l3 [lminus $l1 d]
a b c
```

The following example illustrates the use of **lminus** with the **-exact** option:

```
prompt> set l1 {a a[1] a* b[1] b c}
a a[1] a* b[1] b c

prompt> set l2 [lminus $l1 a*]
{b[1]} b c

prompt> set l3 [lminus -exact $l1 a*]
a {a[1]} {b[1]} b c

prompt> set l4 [lminus -exact $l1 {a[1] b[1]} ]
a a* b c
```

SEE ALSO

```
lreplace(2)
lsearch(2)
```

load_upf

Loads the specified UPF (Unified Power Format) file.

SYNTAX

```
load_upf
   [ -container container_name | -r | -i ]
   [ -scope \instance_path ]
   [ -version version_string ]
   [ -force ]
   [ -strict_check true|false ]
   [ -supplemental supplemental_upf_file_name ]
   [ -target target_type ]
   file_name
```

Data Types

```
container_name string
instance_path string
version_string string
strict_check Boolean
supplemental_upf_file_name string
target_type string
file_name string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

-container *container_name*

Loads the UPF file to the specified container.

-r

Loads the UPF file to the reference container.

-i

Loads the UPF file to the implementation container.

-scope \*instance_path*

Sets the initial scope for the UPF to the named instance.

-version *version_string*

Specifies the version of the UPF file that is being loaded. If the version specified by using the **upf_version** command in the UPF file does not match the version specified by using the *version_string* argument, the command issues a warning.

-force

Loads the UPF file into a container that was previously modified by UPF or the automated setup file for verification (SVF). Using this option is neither safe nor recommended, and is normally not allowed, because previous UPF or automated setup modifications might have invalidated design object name references in the UPF.

-strict_check true|false

Default is **true**. When **false** (as for "Golden UPF" application to a netlist):

- The **upf_name_map_file** variable or **upf_name_map** pragmas in the target design (if any) are used to source map files.
- Missing object errors are suppressed. (If **golden_upf_report_missing_objects** is true, informational messages are reported.)
- ***_logic_*** commands are ignored.

-supplemental *supplemental_upf_file_name*

Specifies a supplemental UPF file to be loaded after the primary one. During the supplemental UPF load (after the primary UPF load), **strict_check false** is ignored, and any name maps or rules used in the primary UPF load are discarded.

-target *target_type*

Specifies the origin of the target design:

- **automatic**: default interpretation, based on **upf_implementation_based_on_file_headers** and/or **upf_implemented_constructs**.
- **rtl**: standard interpretation: no part of the UPF is assumed to be already implemented in the incoming target design. Formality must implement all UPF constructs. **upf_implementation_based_on_file_headers** and **upf_implemented_constructs** are ignored.
- **dc_netlist**: specifies that repeater, isolation, and retention strategies are already implemented in the incoming target netlist, but resulting cells are not connected to their supplies. **upf_implementation_based_on_file_headers** and **upf_implemented_constructs** are ignored.
- **icc_netlist**: specifies that repeater, isolation, and retention strategies and power switches are already implemented in the incoming target netlist, and resulting cells are explicitly connected to their supplies via **connect_supply_net** commands in the incoming UPF. **upf_implementation_based_on_file_headers** and **upf_implemented_constructs** are

ignored.

- **dc_pg_netlist**: as for **dc_netlist**, but also specifies that supplies are already implemented and explicitly connected in the incoming target netlist:

1. an automatic **-reuse** is applied to **create_supply_*** commands;
2. **connect_supply_net** commands are ignored;
3. no automatic supply net connection occurs.

upf_implementation_based_on_file_headers and **upf_implemented_constructs** are ignored.

file_name

Specifies the name of the UPF file to load.

DESCRIPTION

Use this command to load and execute the UPF commands in the specified UPF file, and apply them to the specified container. If a container is not specified, the command applies the UPF commands to the default container. The command reports an error if a container is not specified and there is no default container. To run this command, ensure that the tool is in the setup mode, and the top design in the container is set.

If a UPF command reports an error, the tool stops processing the rest of the UPF file, unless the **sh_continue_on_error** variable is set to *true*.

The tool supports the UPF commands necessary to complete low power design by using the Synopsys low power design flow.

See the Synopsys low power flow documentation and Synopsys SolvNet for more details on the IEEE 1801, also knows as Unified Power Format (UPF), commands supported in the Synopsys flow.

The **load_upf** command returns one of the following:

- `0 to indicate failure`

- 1 to indicate success

EXAMPLES

```
fm_shell (setup)> load_upf myfile.upf
1
fm_shell (setup)>
```

SEE ALSO

"IEEE Std 1801"

ls

Lists the contents of a directory.

SYNTAX

```
string ls [filename ...]

string filename
```

ARGUMENTS

filename

Provides the name of a directory or filename, or a pattern which matches files or directories.

DESCRIPTION

If no argument is specified, the contents of the current directory are listed. For each *filename* matching a directory, **ls** lists the contents of that directory. If *filename* matches a file name, the file name is listed.

EXAMPLES

```
shell> ls *.db *.pt

test1.pt        c1.db         c3.db         c5.db

test2.pt        c2.db         c4.db         c6.db
```

SEE ALSO

cd(2)
pwd(2)

man

Displays the manual pages for Formality shell commands.

SYNTAX

```
str man
[ command ]
```

Data Types

```
command string
```

ARGUMENTS

command

Specifies a Formality shell command.

DESCRIPTION

Use this command to display the manual page for any Formality shell command.

EXAMPLES

This example displays the man page for the **read_db** command:

```
      fm_shell> man read_db
```

SEE ALSO

```
   help(2)
```

match

Matches the implementation design against the reference design based on compare points.

SYNTAX

```
status match
```

DESCRIPTION

When you run the **match** command the tool changes to the match mode and matches the compare points in the implementation design against the compare points in the reference design. After matching is complete, the command reports the results of the match.

You can perform compare point matching incrementally. This is useful when automatic matching fails because of differences in the name and the structure between the reference and implementation designs. If you interrupt a match process, the tool retains the partial matching results. Run the **match** command again to resume matching.

You can issue commands that control matching, such as the **set_compare_rule** or **set_user_match** commands, if unmatched points remain in the reference design when matching completes. Run the **match** command again until all compare points in the reference design are matched. The **verify** command does not match previously matched compare points.

It is not necessary to execute this command before the **verify** command. If matching was not performed, the tool matches the compare points before verification automatically.

Use the **undo_match** command to undo the matched results from the most recent match. Run the **undo_match** command to undo the previous batch of matches.

The **match** command prints a summary of matching results and moves all matched compare points to the unverified state. Use the **report_unmatched_points** and **report_matched_points** commands to generate detailed reports.

Unmatched compare points do not cause errors. However, unmatched or incorrectly matched reference compare points might result in a failed verification.

EXAMPLES

The following example shows all matched reference compare points.

```
fm_shell (setup)> match
Reference design is 'ref:/FMLIB/mif_top'
Implementation design is 'impl:/FMLIB/mif_top'
Status:  Checking designs...
    Warning: Constraint 'CON2' is disabled for contexts above 'ref:/FMLIB/m_adr'
             because it constrains hierarchical design ports. (FM-143)
    Warning: Constraint 'CON1' is disabled for contexts above 'ref:/FMLIB/m_cmd'
             because it constrains hierarchical design ports. (FM-143)
Status:  Building verification models...
Status:  Matching...

**********Matching Results********************

 288 Compare points matched by name
 30 Compare points matched by signature analysis
 0 Compare points matched by topology
 0(31) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black box outputs
-----------------------------------------------

Unmatched Objects                REF        IMPL

-----------------------------------------------

 Registers                       0          31

   Clock-gate LAT                0          31

************************************************
```

The following example shows some remaining unmatched compare points.

```
fm_shell (match)> match
Reference design is 'ref:/WORK/sequenceur'
Implementation design is 'imp:/WORK/SEQUENCEUR'
Status:  Checking designs...
    Info:  0 (4) multiply-driven nets found in reference (implementation) design;
 see formality.log for list.
Status:  Building verification models...
Status:  Matching...

***********Matching Results********************

35 Compare points matched by name
44 Compare points matched by signature analysis
0 Compare points matched by topology
9(9) Unmatched reference(implementation) compare points
0(0) Unmatched reference(implementation) primary inputs, black box outputs
--------------------------------------------------

Unmatched Objects                 REF        IMPL

--------------------------------------------------

 Registers                        9          9

   DFF                            9          9
```

```
      **************************************************
```

SEE ALSO

```
report_unverified_points(2)
report_matched_points(2)
report_unmatched_points(2)
remove_user_match(2)
report_user_matches(2)
set_user_match(2)
setup(2)
undo_match(2)
verify(2)
```

match_eco_regions

Identifies ECO regions.

SYNTAX

```
status match_eco_regions
```

Enabled Shell Modes

Preverify
Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

DESCRIPTION

This command can be issued in preverify, match or verify mode. You must first specify the original reference, original implementation, and the ECO reference. The command will identify ECO regions in the ECO reference and their matching boundaries in the original implementation.

SEE ALSO

```
write_eco_regions
create_eco_patch
set_orig_reference
```

```
set_orig_implementation
set_eco_reference
set_eco_implementation
```

memory

Reports **memory** used by the Formality shell.

SYNTAX

```
int memory
   [-format]
   [-units]
   [-tag]
   [-now]
   [-sub]
```

ARGUMENTS

-format

Formats output with appropriate units.

-units kB | mB | gB

Specify which units to use. The default is kB.

-tag

Reports which parallel section, if any, set the high-water mark.

-now

Reports memory used by the current main process, ignoring child processes that are no longer active and that may have set a higher highwater mark.

-sub

Reports maximum memory used by any subprocess.

DESCRIPTION

Use this command to report memory used by the Formality shell. When using multi-core (set_host_options -max_cores <n>, n > 1), this command reports an upper bound on the maximum high watermark of memory over the life of the Formality session, including child processes, which may be more than the current memory in use. By default the memory is rounded to the nearest kilobyte.

Note that the Linux tools, such as ps, top, lsf, generally do not distinguish private vs. shared memory in a multi-core environment, so they erroneously report memory usage that is higher than actual use, and higher than the numbers reported by Formality.

EXAMPLES

The following example returns memory used by the Formality shell.

```
fm_shell> memory
4880
```

SEE ALSO

parse_proc_arguments

Parses the arguments passed into a Tcl procedure.

SYNTAX

```
string parse_proc_arguments -args arg_list
result_array

list arg_list
string result_array
```

ARGUMENTS

***-args** arg_list*

Specified the list of arguments passed in to the Tcl procedure.

result_array

Specifies the name of the array into which the parsed arguments should be stored.

DESCRIPTION

The **parse_proc_arguments** command is used within a Tcl procedure to enable use of the -help option, and to support argument validation. It should typically be the first command called within a procedure. Procedures that use **parse_proc_arguments** will validate the semantics of the procedure arguments and generate the same syntax and semantic error messages as any application command (see the examples that follow).

When a procedure that uses **parse_proc_arguments** is invoked with the -help option, **parse_proc_arguments** will print help information (in the same style as using **help -verbose**) and will then cause the calling procedure to return. Similarly, if there was any type of error with the arguments (missing required arguments, invalid value, and so on), **parse_proc_arguments** will return a Tcl error and the calling procedure will terminate and return.

If you didn't specify -help, and the specified arguments were valid, the array variable *result_array* will contain each of the argument values, subscripted with the argument name. Note that the argument name here is NOT the names of the arguments in the procedure definition, but rather the names of the arguments as defined using the **define_proc_attributes** command.

The **parse_proc_arguments** command cannot be used outside of a procedure.

EXAMPLES

The following procedure shows how **parse_proc_arguments** is typically used. The argHandler procedure parses the arguments it receives. If the parse is successful, argHandler prints the options or values actually received.

```
proc argHandler args {
  parse_proc_arguments -args $args results
  foreach argname [array names results] {
    echo "  $argname = $results($argname)"
  }
}
define_proc_attributes argHandler -info "argument processor" \
  -define_args \
  {{-Oos "oos help" AnOos one_of_string {required value_help {values {a b}}}}
   {-Int "int help" AnInt int optional}
   {-Float "float help" AFloat float optional}
   {-Bool "bool help" "" boolean optional}
   {-String "string help" AString string optional}
   {-List "list help" AList list optional}}
  {-IDup int dup AIDup int {optional merge_duplicates}}}
```

Invoking argHandler with the -help option generates the following:

```
prompt> argHandler -help
Usage: argHandler    # argument processor
        -Oos AnOos             (oos help:
                                Values: a, b)
        [-Int AnInt]           (int help)
        [-Float AFloat]        (float help)
        [-Bool]                (bool help)
        [-String AString]      (string help)
        [-List AList]          (list help)
```

Invoking argHandler with an invalid option causes the following output (and a Tcl error):

```
prompt> argHandler -Int z
Error: value 'z' for option '-Int' not of type 'integer' (CMD-009)
Error: Required argument '-Oos' was not found (CMD-007)
```

Invoking argHandler with valid arguments generates the following:

```
prompt> argHandler -Int 6 -Oos a
  -Oos = a
  -Int = 6
```

SEE ALSO

define_proc_attributes(2)
help(2)
proc(2)

preverify

Causes Formality to process SVF and enter Preverify mode.

SYNTAX

```
status preverify
```

Enabled Shell Modes

Setup
Match
Verify

DESCRIPTION

This command causes Formality to process SVF and enter Preverify mode. Existing matching and verification results are discarded.

Preverify mode can be used to perform setup operations on post-SVF modified design objects. However, only setup operations that do not modify the design database can be peformed in Preverify mode.

For example, SVF processing may change the name of a register object. If a constant is to be set on this register, its post-SVF processed name must be guessed in Setup mode. Preverify mode alleviates this problem by giving you access to the post-SVF processed register name.

In Preverify mode, executing the **match** command causes Formality to perform compare point matching and switches to Match mode. Executing the **verify** command will cause Formality to switch to Verify mode. Executing **setup** will switch Formality back to Setup mode.

The prompt changes to reflect the current mode, as follows:

```
fm_shell (guide)>
fm_shell (setup)>
fm_shell (preverify)>
fm_shell (match)>
fm_shell (verify)>
```

EXAMPLES

The following example illustrates entering Preverify mode from Setup mode.

```
fm_shell (setup)> preverify
Reference design is 'r:/WORK/top'
Implementation design is 'i:/WORK/top'
Status:  Checking designs...
Status:  Building verification models...
    Status:  Creating views...

MILESTONE: SVF; 171 MB; 26.7964 s; A:1; R:0;
    Status:  Defining RTL constraints...

    Arch      Source      Object ID
    No multipliers match these criteria.


    Status:  Propagating constraints...
    Status:  Starting vtopo managers...
    Status:  Initializing cpoint manager...
    Status:  Identifying black-box resolved nets and marking unread objects...
    Status:  Verification models complete.

***************************** Guidance Summary *****************************
                                          Status
Command                 Accepted   Rejected  Unsupported  Unprocessed  Total
----------------------------------------------------------------------------
uniquify           :          1          0          0          0          1
1
fm_shell (preverify)>
```

SEE ALSO

```
collections(2)
setup(2)
guide(2)
match(2)
verify(2)
```

print_message_info

Prints information about diagnostic messages that have occurred or have been limited.

SYNTAX

```
string print_message_info
   [-ids id_list]
   [-summary]
```

Data Types

```
   id_list      list
```

ARGUMENTS

-ids *id_list*

Specifies a list of message identifiers to report. Each entry can be a specific message or a glob-style pattern that matches one or more messages. If this option is omitted and no other options are given, then all messages that have occurred or have been limited are reported.

-summary

Generates a summary of error, warning, and informational messages that have occurred so far.

DESCRIPTION

The **print_message_info** command enables you to print summary information about error, warning, and informational messages that have occurred or have been limited with the **set_message_info** command. For example, if the following message is generated, information about it is recorded:

Error: unknown command 'wrong_command' (CMD-005)

It is useful to be able to summarize all recorded information about generated diagnostic messages. Much of this can be done using the **get_message_info** command, but you need to know a specific message ID. By default, **print_message_info** summarizes all of the information. It provides a single line for each message that has occurred or has been limited, and one summary line that shows the total number of errors, warnings, and informational messages that have occurred so far. If an *id_list* is given, then only messages matching those patterns are displayed. If **-summary** is given, then a summary is displayed.

Using a pattern in the *id_list* is intended to show a specific message prefix, for example, "CMD*". Note that this does not show all messages with that prefix. It shows only the messages that have occurred or have been limited.

EXAMPLES

The following example uses **print_message_info** to show a few specific messages:

```
prompt> print_message_info -ids [list "CMD*" APP-99]

Id               Limit    Occurrences   Suppressed
----------------------------------------------------
CMD-005              0              7            2
APP-99               1              0            0
```

At the end of the session, you might want to generate some information about a set of interesting messages, such as how many times each occurred (which includes suppressions), how many times each was suppressed, and whether a limit was set for any of them. The following example uses **print_message_info** to get this information:

```
prompt> print_message_info

Id               Limit    Occurrences   Suppressed
----------------------------------------------------
CMD-005              0             12            0
APP-027            100            150           50
APP-99               0              1            0

Diagnostics summary: 12 errors, 150 warnings, 1 informational
```

Note that the suppressed count is not necessarily the difference between the limit and the occurrences, since the limit can be dynamically changed with the **set_message_info** command.

SEE ALSO

```
get_message_info(2)
set_message_info(2)
suppress_message(2)
```

print_suppressed_messages

Displays an alphabetical list of message IDs that are currently suppressed.

SYNTAX

```
string print_suppressed_messages
```

ARGUMENTS

The **print_suppressed_messages** command has no arguments.

DESCRIPTION

The **print_suppressed_messages** command displays all messages that you suppressed using the **suppress_message** command. The messages are listed in alphabetical order. You only can suppress informational and warning messages. The result of **print_suppressed_messages** is always the empty string.

EXAMPLES

The following example shows the output from the **print_suppressed_messages** command:

```
prompt> print_suppressed_messages
No messages are suppressed

prompt> suppress_message {XYZ-001 CMD-029 UI-1}

prompt> print_suppressed_messages
The following 3 messages are suppressed:
```

```
            CMD-029, UI-1, XYZ-001
```

SEE ALSO

```
suppress_message(2)
unsuppress_message(2)
```

printenv

Prints the value of each environment variable.

SYNTAX

```
printenv
   [pattern]
```

Data Types

```
pattern string
```

ARGUMENTS

pattern

Specifies a single variable to print.

DESCRIPTION

Use this command to print the values of the environment variables inherited from the parent process or set in the tool by using the **set_unix_variable** command. Unless a variable is specified, the command prints the values of all environment variables.

To retrieve the value of a single environment variable, you can also use the **get_unix_variable** command.

EXAMPLES

These examples show the output of the **printenv** command.

```
    fm_shell> printenv SHELL
    /bin/csh
    fm_shell> printenv EDITOR
    emacs
```

SEE ALSO

```
get_unix_variable(2)
set_unix_variable(2)
```

printvar

Prints the values of one or more variables.

SYNTAX

```
printvar
   [ -application ]
   [ -user_defined ]
   [ pattern ]
```

Data Types

pattern

ARGUMENTS

-application

Prints the values of the application variables.

-user_defined

Prints the values of the user-defined variables.

pattern

Prints the value of the specified *pattern* (variable).

DESCRIPTION

Use this command to print the values of one or more patterns.

If a *pattern* is not specified, the command prints out the values of all the variables, both Formality and user-defined.

EXAMPLES

This example prints the values of all the variables.

```
fm_shell> printvar
```

This example prints the values of all the variables that start with sh*.

```
fm_shell> printvar sh*
sh_arch                    ="sparc"
sh_continue_on_error       = "0"
sh_enable_page_mode        = "false"
sh_product_version         = "1997.01-development"
sh_source_uses_search_path = "false"
```

This example prints the value of the variable search_path.

```
fm_shell> printvar search_path
search_path = ".  /designs/newcpu/v1.6  /lib/cmos"
```

SEE ALSO

proc_args

Displays the formal parameters of a procedure.

SYNTAX

```
str proc_args
   <proc_name>
```

Data Types

```
<proc_name> string
```

ARGUMENTS

proc_name

Specifies the name of the procedure.

DESCRIPTION

This command displays the names of the formal parameters of a user defined procedure.

This command is essentially a synonym for the Tcl builtin command **info** with the **args** argument.

EXAMPLES

This example shows the output of **proc_args** for a simple procedure.

```
fm_shell> proc plus {a b} { return [expr $a + $b] }
fm_shell> proc_args plus
a b
fm_shell> info args plus
a b
```

SEE ALSO

```
info(2)
proc(2)
proc_body(2)
```

proc_body

Displays the body of a procedure.

SYNTAX

```
proc_body
   proc_name
```

Data Types

```
proc_name string
```

ARGUMENTS

proc_name

Specifies the name of the procedure.

DESCRIPTION

This command displays the contents of a user-defined procedure.

This command is essentially a synonym for the Tcl builtin command **info** with the **body** argument.

EXAMPLES

This example shows the output of **proc_body** for a simple procedure.

```
fm_shell> proc plus {a b} { return [expr $a + $b] }
fm_shell> proc_body plus
 return [expr $a  + $ b]
```

SEE ALSO

```
info(2)
proc(2)
proc_args(2)
```

query_objects

Searches for and displays objects in the database.

SYNTAX

```
string query_objects
   [-verbose]
   [-truncate elem_count]
   [-class class_name]
   object_spec
```

Data Types

```
class_name          string
elem_count          int
object_spec         list
```

ARGUMENTS

-verbose

Displays the class of each object found. By default, only the name of each object is listed. With this option, each object name is preceded by its class (see the EXAMPLES).

-truncate *elem_count*

Truncates display to *elem_count* elements. By default, up to 100 elements display. To see more or less elements, use this option. To see all elements, set the *elem_count* value to 0.

-class *class_name*

For elements in the *object_spec* that are not collections, this is the class used when searching the database for objects which match the element. Valid classes are application-specific.

object_spec

Provides a list of objects to find and display. Each element in the list is either a collection or a pattern which will match some objects in the database. Patterns are explicitly searched for in the database with class *class_name*.

DESCRIPTION

The **query_objects** command finds and displays objects in the application's runtime database. The command does not have a meaningful return value; it displays the objects found and returns an empty string.

The *object_spec* is a list containing collections and/or object names. For elements of the *object_spec* that are collections, **query_objects** simply displays the contents of the collection.

For elements of the *object_spec* that are names (or contain wildcard patterns), the **query_objects** command searches the database for objects of the class specified by the *class_name* option. Note: The **query_objects** command does not have a predefined implicit order of classes for which searches are initiated. If you do not specify the *class_name* option, only those elements that are collections are displayed. Messages are displayed for the other elements (see EXAMPLES).

To control the number of elements displayed, use the *-truncate* option. If the display is truncated, you see the ellipsis (...) as the last element. Note that if the default truncation occurs, a message displays showing the total number of elements that would have displayed (see EXAMPLES).

> NOTE: The output from the **query_objects** command looks similar to the output from any command that creates a collection; however, the result of the **query_objects** command is always an empty string.

EXAMPLES

These following examples show the basic usage of the **query_objects** command.

```
prompt> query_objects [get_cells o*]
{r:/WORK/top/or1 r:/WORK/top/or2 r:/WORK/top/or3}
prompt> query_objects -class cell U*
{r:/WORK/top/U1 r:/WORK/top/U2}
prompt> query_objects -verbose -class cell [list U* [get_nets n1]]
{{cell r:/WORK/top/U1} {cell r:/WORK/top/U2} {net r:/WORK/top/n1}}
```

When you omit the **-class** option, only those elements of the *object_spec* that are collections generate output. The other elements generate error messages.

```
prompt> query_objects [list U* [get_nets n1] n*]
Error: No such collection 'U*' (SEL-001)
Error: No such collection 'n*' (SEL-001)
{r:/WORK/top/n1}
```

When the output is truncated, you get the ellipsis at the end of the display. For the following example, assume the default truncation is 5 (it is actually 100).

```
prompt> query_objects [get_cells o*] -truncate 2
{r:/WORK/top/or1 r:/WORK/top/or2 ...}
prompt> query_objects [get_cells *]
{r:/WORK/top/or1 r:/WORK/top/or2 r:/WORK/top/or3 r:/WORK/top/U1 r:/WORK/top/U2 ...}
Output truncated (total objects 126)
```

SEE ALSO

```
collections(2)
get_cells(2)
get_clocks(2)
get_designs(2)
get_generated_clocks(2)
get_lib_cells(2)
get_lib_pins(2)
get_libs(2)
get_nets(2)
get_path_groups(2)
get_pins(2)
get_ports(2)
get_qtm_ports(2)
get_timing_paths(2)
```

quit

Exits the Formality shell.

SYNTAX

```
quit
```

DESCRIPTION

This command exits the Formality shell and closes the current Formality session.

EXAMPLES

The following example exits the Formality shell.

```
fm_shell> quit
```

SEE ALSO

```
exit(2)
```

read_container

Reads a container into the Formality environment.

SYNTAX

```
read_container
   [ -container container_name | -r | -i ]
   [ -replace ]
   file_name
```

Data Types

```
   container_name string
   file_name string
```

ARGUMENTS

-container *container_name*

Specifies the container into which files are read.

-r

Reads container data into the cureent reference container.

-i

Reads container data into the current implementation container.

-replace

Overwrites an existing container. This option is required if the current container is not empty.

file_name

Specifies the name of the file containing the saved container.

DESCRIPTION

This command reads a container into the Formality environment. The file that is read in must be written by Formality.

If the container being read in and the current container have the same name, the command reports an error and does not load the container. If the two containers do not have the same name, the command creates the container, reads the information into it, and establishes it as the current container.

To specify a unique container name, use the **-container** option. The *-replace* option replaces the contents of the existing container with the contents of the new container.

Containers that are read from files, written without using the **I-pre_set_top** option, marked "read-only," and operations that could modify their contents are not allowed.

You cannot run the following commands on a read-only container, or on objects within a read-only container:

- **set_top**
- **read_db**
- **read_verilog**
- **read_vhdl**
- **remove_design**
- **remove_design_library**
- **remove_library**
- **remove_object**
- **rename_object**
- **set_direction**
- **uniquify**

The **read_container** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example creates a new container named spec_2.

```
fm_shell> read_container -container spec_2 ref.fsc
Loading container file '/u/formality/spec_2.fsc'
Created container 'spec_2'
Current container set to 'spec_2'
1
fm_shell>
```

The following example reads the information from a previously saved container back into an existing container.

```
fm_shell> read_container -replace ref.fsc
Loading container file '/u/formality/ref.fsc'
Reference design is no longer set
1
fm_shell>
```

SEE ALSO

```
report_containers(2)
write_container(2)
```

read_db

Reads technology libraries or designs in the .db format.

SYNTAX

```
read_db
   [ -container container_name | -r | -i ]
   [ -libname libname ]
   [ -technology_library]
   [ -merge ]
   [ -replace_black_box ]
   file_names
```

Data Types

```
   container_name string
   libname string
   file_names string
```

ARGUMENTS

-container *container_name*

Specifies the container into which the data is read and sets it as the current container.

-r

Reads data into the default reference container and sets it as the current container.

-i

Reads data into the default implementation container and sets it as the current container.

-technology_library

Specifies that the file is a technology library.

-libname *libname*

Specifies the design library or the technology library into which the designs or cells are read.

-merge

Replaces black box cells and test cells in the specified library with those that are read in from the .db format library. This option also adds cells if there are no corresponding cells already in the specified existing library.

-replace_black_box

Replaces black box cells and test cells in the specified library with those that are read in from the .db format library.

file_names

Specifies one or more files to read. Specify Synopsys database files.

DESCRIPTION

Use this command to read one or more Synopsys internal database (.db) designs or technology libraries into a container. The **read_db** command recognizes whether a file represents a design or a technology library. The command overwrites existing designs in the container with the designs that are read. Designs linked to the overwritten files are unlinked.

Unless you specify the name of the design or technology library, the command uses the default design library named *WORK* or the default technology library named *TECH_WORK*.

To merge the specified existing library and the .db format library being read in, use the *-merge* option. Merging replaces black box and test cells in the existing library with those read in from the database. Merging also adds cells if there are no corresponding cells in the existing library.

To replace black box and test cells in the specified existing library with those read in from the database without merging the two libraries, use the *-replace_black_box* option.

The following list describes how to use the **read_db** command.

- **Reading designs in .db format into the current container** - To read a design library in .db format into the current container, use the **read_db** command. Do not use the *-container* option. If you use the *-container* option and you have not established a current container, the command reports an error. Use the following command:

```
read_db file
```

- **Reading a design library in .db format into a specific container** - To read a design library in the .db format into a specific container, use the *-container* option. If you use the *-container* option and the container does not exist, the command creates the container and establishes it as the current container. Use the following command:

```
read_db -container c_name file
```

- **Reading shared database technology libraries** - To read a shared database technology library, use the **read_db** command. Do not use the *-container* option, which reads the library into all currently open containers and all subsequently opened containers. Use the following command:

```
read_db file
```

- **Reading unshared database technology libraries** - To read a database technology library into a specific container, use the *-container* option. If you use the *-container* option and the container does not exist, the command creates the container and establishes it as the current container. Use the following command:

```
read_db -container c_name file
```

- **Renaming a design library** - To rename a design library, specify the *-libname* option. By default, designs are read into the default design library called *WORK*. Use the following command:

```
read_db -libname my_design_lib file
```

- **Renaming a technology library** - To rename a technology library, specify the *-libname* option. By default, cells are read into the default technology library called *TECH_WORK*.

```
read_db -libname my_tech_lib file
```

The **read_db** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

Special Behavior in the Library Verification Mode

In the library verification mode, use the **read_db** command to read all libraries in .db format into a specified container. The container library names are handled internally by the tool. The only options that remain valid in the library verification mode are *-r* and *-i*.

```
fm_shell> read_db -r mylib.db
Loading db file 'mylib.db'
Current container set to 'r'
Total cells found: 2
1
```

EXAMPLES

This example reads a design library in .db format. In this case, the command creates the container by using the *-container* option. Note the design files are read into a default design library named *WORK* and the newly created container is established as the current container.

```
fm_shell> read_db -c ref post_scan.db
Loading db file 'post_scan.db'
No target library specified, default is WORK
Created container 'ref'
Current container set to 'ref'
1
fm_shell>
```

This example reads a design library in .db format into the current container. Here, the *-libname* option is used to rename the design library to *my_lib*.

```
fm_shell> create_container ref
Created container 'ref'
```

```
Current container set to 'ref'
1
fm_shell> read_db -libname my_lib post_scan.db
Loading db file 'post_scan.db'
1
fm_shell> report_design_libraries

Number of
Designs     Design Library
---------   --------------
       10      ref:/my_lib
1
```

This example merges the contents of the specified library in .db format into the shared technology library *my_lib*, replacing black box and test cells that are already in the library *my_lib*.

```
fm_shell> read_db -libname my_lib -replace_black_box resolve.db
Loading db file 'resolve.db'
Loading new version of design 'LS1P' into shared technology library 'my_lib'
Loading new version of design 'LS1' into shared technology library 'my_lib'
1
fm_shell>
```

This example merges the contents of the specified library in .db format into the technology library *new_lib* that is not shared, replacing black box and test cells and adding cells when there is no corresponding cell already in the library *new_lib*.

```
fm_shell> read_db -c ref -libname new_lib -merge new_versions.db
Loading db file 'new_versions.db'
1
fm_shell>
```

This example reads a shared technology library (all containers get the library). Note that when you create subsequent containers, the command loads the technology library automatically into the new container.

```
fm_shell> read_db lca500k.db
Loading db file 'lca500k.db'
1
fm_shell>
```

SEE ALSO

```
report_designs(2)
report_design_libraries(2)
report_libraries(2)
```

read_ddc

Reads Synopsys logical database in the .ddc format.

SYNTAX

```
status read_ddc
   [ -container container_name | -r | -i]
   [ -libname libname]
   [ -technology_library]
   [ -merge]
   [ -replace_black_box]
   [ -block_abstraction block_names]
   file_names
```

Data Types

```
container_name string
libname string
file_names string
block_names string
```

ARGUMENTS

-container *container_name*

Specifies the container into which the file is to be read and sets it as the current container.

-r

Reads the file into the default reference container and sets it as the current container.

-i

Reads the file into the default implementation container and sets it as the current container.

-technology_library

Indicates that the specified file as a technology library.

-libname *libname*

Specifies the design library or the technology library into which the designs in .ddc format or cells are read.

-merge

Replaces black box cells and test cells in the existing library with those read in from the designs or cells. This option adds cells if there are no corresponding cells already in the existing library.

-replace_black_box

Replaces black box cells and test cells in the specified existing library with those read in from the designs or cells.

block_names

Specifies a list of block abstraction module or design names.

file_names

Specifies one or more files to read. Specify one or more Synopsys designs in .ddc format.

DESCRIPTION

This command reads one or more design files that are in .ddc format into the specified container. When existing designs are present in the destination container, the tool overwrites existing designs with the designs that are read. Designs linked to the overwritten files are unlinked.

By default, the tool names the design library *WORK*, or *TECH_WORK* if it is a technology library.

If you specify the **-merge** option for a .ddc format design,

- The contents of the file are merged into the specified existing library
- Black box and test cells in the existing library are replaced with those read in from the database
- Cells are added evef if there are no corresponding cells in the existing library.

The **-replace_black_box** option only replaces black box and test cells in the named existing library with those read in from the database.

The **-block_abstraction** switch allows list of designs to be read as interface logic for Transparent Interface Optimization(TIO) flow.

The following list describes the use of the **read_ddc** command:

- **Reading .ddc format designs into the current container** - To read designs in .ddc format into the current container, use the **read_ddc** command. Do not use the **-container** option. The syntax to read .ddc format designs into the current directory is:

  ```
  read_ddc file_list
  ```

- **Reading .ddc format designs into a specific container** - To read .ddc format designs into a specific container, use the **-container** option. If you use the **-container** option and the container does not exist, the tool creates the container and establishes it as the current container. The syntax to read .ddc format designs into a specific directory is:

```
   read_ddc -container container_name
```

- **Renaming a design library** - To rename a design library, use the **-libname** option. By default, designs are read into the default design library named *WORK*. The command syntax to rename a design library is:

```
   read_ddc -libname libname
```

EXAMPLES

The following example reads a .ddc format design. The command creates a container by using the **-container** option. Note that the design files are read into a default design library named *WORK* and the newly created container is established as the current container.

```
prompt> read_ddc -c ref post_scan.ddc
Loading ddc file 'post_scan.ddc'
Loaded 22 designs.
< ... list of designs omitted ... >
No target library specified, default is WORK
Created container 'ref'
Current container set to 'ref'
1
```

This example reads a .ddc format design into the current container. The **-libname** option is used to rename the design library to *my_lib*.

```
prompt> create_container ref
Created container 'ref'
Current container set to 'ref'
1
prompt> read_ddc -libname my_lib \
post_scan.ddc
Loading ddc file 'post_scan.ddc'
Loaded 22 designs.
< ... list of designs omitted ... >
1
prompt> report_design_libraries
Number of
Designs     Design Library
---------   --------------
       22      ref:/my_lib
1
```

SEE ALSO

```
report_designs(2)
report_design_libraries(2)
report_libraries(2)
```

read_fsm_states

Reads finite state machine (FSM) states into the Formality environment.

SYNTAX

```
read_fsm_states
   file
   [ designID ]
```

Data Types

```
designID string
```

ARGUMENTS

file

Specifies the file containing the FSM states.

designID

Specifies the design with FSM states you are defining.

DESCRIPTION

This command reads FSM information, enabling the tool to correctly process FSM designs that have different state encodings. For the tool to verify one FSM against another, the FSM designs must use the same state names.

The specified file must be generated by using the **report_fsm** command in Design Compiler or be a text file that follows these rules:

- Must contain one-word directives that start with the period (.) character.
- Single-line comments must start with the pound character (#). White space must precede the pound

character. The newline character is considered white space.

- The file must contain the ".state_vector" directive followed by a single or multiline list of FSM flip-flop names. This list defines the FSM state vectors.
- The file must contain the ".encoding" directive followed by a list of state names and their binary encodings. For example, the directive could be followed by this line:

  ```
  RESET_STATE 2#100 FINAL_STATE 2#011;
  ```

For each state name and encoding pair, the number and order of binary digits must match the number and order of flip-flops in the state vectors. You can separate binary digits with an underscore character "_". You can specify the number base as binary, octal, decimal, or hexadecimal.

The following example specifies FSM states. It names two flip-flops that hold states and defines four state names and their encodings. Notice the various methods by which you can specify the numbering system.

```
CURRENT_STATE_reg[1]
CURRENT_STATE_reg[0]

SO ^B0_0
S1 16#2
S2 ^O1
S3 8#3
```

You can specify the design into which states are read by specifying the *designID* argument. If you do not specify a design ID, the command reads the states into the current design. If the command cannot match the FSM state information in the file to the design, it reports the following error, where *name* is the flip-flop that could not be found:

```
Error:  No Flip-Flop name in the design.
```

The **read_fsm_states** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example reads FSM states into the current design. The states are located in a file named *state_info* in the current working directory.

```
fm_shell> read_fsm_states state_info
1
fm_shell>
```

The following example reads FSM states into the design *ref:/WORK/CORE/FSM*. The states are located in a file named *state_info*.

```
fm_shell> read_fsm_states $HOME/misc/state_info \
ref:/WORK/CORE/FSM
1
```

SEE ALSO

```
report_fsm(2)
set_fsm_encoding(2)
set_fsm_state_vector(2)
```

read_milkyway

Reads in a design from the Milkyway design library.

SYNTAX

```
read_milkyway
   [ -container container_name | -r | -i ]
   [ -libname libname ]
   [ -technology_library ]
   [ -version version_number ]
   [ -no_pg ]
   [ -block_abstraction block_names]
   -cell_name mw_cell_name
   mw_db_path
```

Data Types

```
container_name string
libname string
version_number integer
mw_cell_name string
mw_db_path string
```

ARGUMENTS

-container *container_name*

Specifies the container into which the data is read. This option also establishes the *container_name* as the current container.

-r

Sets the default reference container as the current container and reads the specified data into it.

-i

Sets the default implementation container as the current container and reads the specified data into it.

-libname *libname*

Reads designs or cells into the specified design library or the technology library.

-technology_library

Specifies that the file that is read in is a technology library.

-version *version_number*

Specifies the version of the design file that needs to be read. For example, design_lib/CEL/design1:1 and design_lib/CEL/design1:2, here, 1 and 2 after \':\' are the version numbers. This is an optional argument, and the default is the latest version.

-no_pg

Specifies that the Milkyway design and the technology library cells are not to be linked to power and ground pins. This is optional, and the default behavior is to link to pg_pin cell versions. Use this option when reading Milkyway designs generated by Design Compiler because usually the Milkyway design it generates does not have power and ground pin connections.

-block_abstraction *block_names*

Specifies a list of block abstraction modules or designs to be read as interface logic for the Transparent Interface Optimization (TIO) flow.

-cell_name *mw_cell_name*

Specifies the design file name to be read. For example, there are design files under the CEL view in the Milkyway design library design_lib:

```
\'design_lib/CEL/design1_pre_route:1\',
\'design_lib/CEL/design1_post_route:2\'
```

The *design1_pre_route* or *design1_post_route* is *cell_name* arguments. Do not include the version number in this argument.

mw_db_path

Specifies the name and path of the Milkyway design library.

DESCRIPTION

This command reads the design from the Milkyway database on the disk into memory. The hierarchical netlist data is read in. Synthesis constraints or physical data such as floorplan data or placement and routing information are not read in.

If you do not specify the name of the design library or technology library, the command uses the default name - *MW_DESIGN_LIBRARY*.

The **read_milkyway** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

This shows the typical Milkyway database directory hierarchy.

```
my_design_lib/                    (Milkyway design library)
my_design_lib/CEL/                (CEL view)
my_design_lib/CEL/design1_john:1 (John's CEL data for design1)
my_design_lib/CEL/design1_mike:1 (Mike's CEL data for design1)
```

This example reads John's CEL data for design1 to reference container and default library name MW_DESIGN_LIBRARY.

```
read_db -tech tech_link_lib.db
read_milkyway -r -cell_name design1_john my_design_lib
set_top -a
```

This example reads Mike's CEL data from design1 into container my_cont and library my_design_lib.

```
read_db -tech tech_link_lib.db
read_milkyway -con my_cont -libname my_design_lib -cell_name design1_mike my_design_lib
set_top -a
```

SEE ALSO

```
set_top(2)
mw_logic0_net(3)
mw_logic1_net(3)
```

read_power_model

Reads Formality power model files.

SYNTAX

```
read_power_model
   [ -container container_name | -r | -i]
   [ -libname libname]
   file_names
```

Data Types

```
container_name string
libname string
file_names string
```

ARGUMENTS

-container *container_name*

Specifies the container into which the file is to be read and sets it as the current container.

-r

Reads the file into the default reference container and sets it as the current container.

-i

Reads the file into the default implementation container and sets it as the current container.

-libname *libname*

Specifies the design library into which the designs in power model format are read.

file_names

Specifies one or more files to read.

DESCRIPTION

The **read_power_model** command reads one or more design files that are in the Formality power model format into the specified container. Existing designs in the destination container are overwritten and designs linked to the overwritten files are unlinked.

The following list describes the use of the **read_power_model** command:

- **Reading .fpm format designs into the current container** - To read designs in the .fpm format into the current container, use the **read_power_model** command. Do not use the **-container** option. The syntax to read .fpm format designs into the current container is,

  ```
  read_power_model file_list
  ```

- **Reading .fpm format designs into a specific container** - To read designs in the .fpm format into a specific container, use the **-container** option. If you use the **-container** option and the specified container does not exist, the tool creates the container and sets it as the current container. The syntax to read .fpm format designs into a specific directory is:

  ```
  read_power_model -container container_name file_list
  ```

- **Specifying a design library** - To specify a design library, use the **-libname** option. The command syntax to read power models into a specified a design library is:

  ```
  read_power_model -libname libname file_list
  ```

EXAMPLES

The following example shows how to read in a Formality power model. The command creates a container using the **-container** option. Note that the design files are read into a default power model library and the newly created container is set as the current container.

```
fm_shell> read_power_model -container ref sub.fpm
Created container 'ref'
Current container set to 'ref'
Info:  Loaded model for design 'sub'.
1
```

The following example shows how to read a Formality power model into the current container. The **-libname** option is used to specify a design library named *my_lib*.

```
fm_shell> create_container ref
Created container 'ref'
Current container set to 'ref'
1
fm_shell> read_power_model -libname my_lib sub.fpm
Info:  Loaded model for design 'sub'.
1
```

SEE ALSO

```
write_power_model(2)
```

read_sverilog

Reads one or more SystemVerilog files.

SYNTAX

```
status read_sverilog
   [ -container container_name | -r | -i ]
   [ -libname libname ]
   [ -work_library libname ]
   [ -uses design_libs_list]
   [ -L design_libs_list]
   [ -technology_library ]
   [ -f VCS_option_file ]
   [ -F VCS_option_file ]
   [ -vcs "VCS options" ]
   [ -define define ]
   [ -3.1a | -05 | -09 | -12 ]
   [ -extra_library_cells cell_list ]
   [ filenames ]
```

Data Types

```
container_name string
libname string
VCS_option_file string
"VCS options" string
define string
design_list string
filenames string
```

ARGUMENTS

-container *container_name*

Specifies the container into which the SystemVerilog designs or cell descriptions are read. The specified container is set as the current container.

-r

Sets the default reference container as the current container and reads the SystemVerilog designs or cell descriptions into it.

-i

Sets the default implementation container as the current container and reads the SystemVerilog designs or cell descriptions into it.

-libname *libname*

Names the design library or technology library into which SystemVerilog designs or cell descriptions are read. You can specify any alphanumeric string for the *libname* argument.

-work_library *library_name*

Same behavior as the *-libname* option.

-technology_library

Specifies that the file that is read in is a technology library. When you use this option, specify a container by using the *-r*, *-i*, or *-container* option.

-uses *design_libs_list*

The list of libraries used to find the references if Formality could not resolve the reference from working library. Libraries from -uses liblist are used only if parent library doesn't have searched design. If the design is not found in library list of **-uses**, search is continued using default search order.

-L *design_libs_list*

The list of libraries used to find the references. Libraries from -L liblist are used even if parent library does have design/interface. If the reference is not found in library list of **-L**, search is canceled and error message is issued.

-f *VCS_option_file*

Specifies that the file is a VCS option file, which contains VCS options or Verilog files.

-F *VCS_option_file*

Same as the -f option but allows you to specify a path to the file and the source files listed in the file do not have to be absolute pathnames.

-vcs *VCS options*

Specifies that the string enclosed within quotation marks is VCS options or Verilog files.

-define *define*

Sets the define option before reading the first SystemVerilog file. The *define* argument might consist of a single value or a space-delimited list of values enclosed within braces.

-3.1a | -05 | -09 | -12

Specifies the standard to use for language interpretation. The default is -12, which specifies the IEEE Standard 1800-2012. Note that setting the *hdlin_sv_packages* variable to "none" specifies the "-3.1a" standard.

-extra_library_cells *cell_list*

Specifies the list of design names(may be unused) to be elaborated. If called before set_top, the cells in cell_list will be elaborated during the subsequent set_top command even if they are not included in the top design hierarchy. This is in addition to the normal behavior of read_verilog. If the cells in cell_list already exist then they will be overwritten, which is the current behavior of read_sverilog. If called after set_top in setup mode, or anytime in match or verify mode, the cells in cell_list (and only those designs) will be elaborated as part of the read_sverilog command. If the library already exists, it will only be possible to add designs to it; that is, it will only be possible to elaborate designs that have not previously been elaborated. It will not be possible to overwrite any existing design. Attempting to overwrite an existing design will result in an error message.

file_list

Specifies one or more SystemVerilog files to read.

DESCRIPTION

Use this command to read one or more SystemVerilog files. Designs are read into design libraries, and Verilog library cell descriptions are read in as technology libraries. By default, the tool places designs into the default design library named *WORK*, and cell descriptions into the default technology library named *TECH_WORK*. The command, by default, detects if files are in compressed gzip format and reads them.

If a specified library does not exist, the tool creates it. If the library exists, it overwrites the existing designs or cell descriptions with the designs or cell descriptions that are read. Designs linked to any overwritten designs are unlinked.

The following list describes how to use the **read_sverilog** command.

- **Reading designs into a specific container** - To read designs into a specific container, do not use the *-technology_library* option. Instead use the *-r*, *-i*, or *-container* options. If the specified container does not exist, the tool creates the container and sets it as the current container. The syntax to use the command is:

  ```
  read_sverilog  -container c_name file
  ```

- **Reading unshared technology libraries** - To read a technology library into a specific container, use the *-technology_library* option and the *-r*, *-i*, or *-container* options. If the specified container does not exist, the tool creates the container and sets it as the current container. The syntax to use the command is:

  ```
  read_sverilog  -technology_library -container c_name tech_file
  ```

- **Renaming a technology library** - To rename a technology library, specify the *-libname*, or the *-work_library*, and *-technology_library* options along with the *-r*, *-i*, or *-container* options. By default, the command reads cell descriptions into the default technology library named *TECH_WORK*. The syntax to use the command is:

  ```
  read_sverilog -technology_library -r -libname my_tech_lib tech_lib
  ```

- **Renaming a design library** - To rename a design library, specify the *-libname* or *-work_library* option and do not use the *-technology_library* option. By default, the command reads designs into the default design library named *WORK*. The syntax to use the command is:

  ```
  read_sverilog  -libname my_design_lib file
  ```

- **VCS options** - The **read_sverilog** command supports some VCS options: *-v*, *-y*, *-f*, *+define*, *+libext* and *+incdir*. Some VCS options are irrelevant, therefore recognized but ignored: *-P* and all the other '+' options. The syntax of the command is:

```
   read_sverilog  -vcs "VCS options"
   read_sverilog  -f VCS_option_file
   read_sverilog  -F VCS_option_file
```

Special Behavior in the Library Verification Mode:

The library_verification command preprocesses all Verilog library cells into specified container. The container library names are handled internally by the tool. The only options that are valid in the library verification mode are *-r*, *-i*, and *-technology_library*.

```
fm_shell> read_sverilog -r vlg_cells.v
Loading verilog file 'vlg_cells.v '
Preprocessing library 'vlg_cells.v' ...
Total cells found: 5
1
```

EXAMPLES

This example reads a SystemVerilog design into the Formality environment. In this case, the *-container* option creates the container *ref*. The design files are read into a default design library named *WORK*, and the newly created container is established as the current container.

```
fm_shell> read_sverilog -c ref p0.sv
Loading verilog file '/u/formality/designs/p0.sv'
No target library specified, default is WORK
Created container 'ref'
Current container set to 'ref'
1
fm_shell> report_design_libraries

Number of
Designs     Design Library
---------  --------------
    1        ref:/WORK
1
fm_shell>
```

This example reads a SystemVerilog design into the current container. Here, the *-libname* option renames the design library to *my_lib*.

```
fm_shell> read_sverilog -libname my_lib p0.sv
Loading verilog file '/u/formality/designs/p0.sv'
1
fm_shell> report_design_libraries -short
Number of
Designs      Design Library
---------    --------------
    1        ref:/my_lib
1
fm_shell>
```

This example reads a technology library into container 'ref'.

```
fm_shell (setup)> read_sverilog -technology_library -container ref lib.v
```

```
Loading verilog file '/u/formality/libraries/lib.v '
No target library specified, default is TECH_WORK
Created container 'ref'
Current container set to 'ref'
1
fm_shell (setup)> report_libraries -short

Number of
Cells        Shared    Technology Library
---------    ------    ------------------
     83       Yes      GTECH

     83       Yes      i:/GTECH

     83       Yes      r:/GTECH

     83       Yes      ref:/GTECH
      1       No       ref:/TECH_WORK

1

fm_shell> create_container impl
Created container 'impl'
Current container set to 'impl'
1
fm_shell> report_libraries -short
Number of
Cells       Shared    Technology Library
---------   ------    ------------------
    1       Yes       TECH_WORK
    79      Yes       gtech

    1       Yes       impl:/TECH_WORK
    79      Yes       impl:/gtech
    1       Yes       ref:/TECH_WORK
    79      Yes       ref:/gtech
1
fm_shell>
```

SEE ALSO

```
read_verilog(2)
report_designs(2)
report_design_libraries(2)
report_libraries(2)
```

read_verilog

Reads one or more Verilog files.

SYNTAX

```
status read_verilog
   [ -container container_name | -r | -i ]
   [ -libname libname ]
   [ -work_library libname ]
   [ -netlist ]
   [ -uses design_libs_list]
   [ -L design_libs_list]
   [ -technology_library ]
   [ -f VCS_option_file ]
   [ -F VCS_option_file ]
   [ -vcs "VCS options" ]
   [ -define define ]
   [ -95 | -01 | -05 ]
   [ -extra_library_cells cell_list ]
   [ filenames ]
```

Data Types

```
container_name string
libname string
VCS_option_file string
"VCS options" string
define string
design_list string
filenames string
```

ARGUMENTS

-container *container_name*

Specifies the container into which the Verilog designs or cell descriptions are read. The specified container is set as the current container.

-r

Sets the default reference container as the current container and reads the Verilog designs or cell descriptions into it.

-i

Sets the default implementation container as the current container and reads the Verilog designs or cell descriptions into it.

-libname *libname*

Names the design library or technology library into which Verilog designs or cell descriptions are read. You can specify any alphanumeric string for the *libname* argument.

-work_library *library_name*

Same behavior as the *-libname* option.

-netlist

Identifies the file as a Verilog structural netlist. This option can reduce the time required to read the file. Use the -95 or the -01 options to select the IEEE Standard Verilog.

-uses *design_libs_list*

The list of libraries used to find the references if Formality could not resolve the reference from working library. Libraries from -uses liblist are used only if parent library doesn't have searched design. If the design is not found in library list of **-uses**, search is continued using default search order.

-L *design_libs_list*

The list of libraries used to find the references. Libraries from -L liblist are used even if parent library does have design/interface. If the reference is not found in library list of **-L**, search is canceled and error message is issued.

-technology_library

Specifies that the file that is read in is a technology library. When you use this option, specify a container by using the *-r*, *-i*, or *-container* options.

-f *VCS_option_file*

Specifies that the file is a VCS option file, which contains VCS options or Verilog files.

-F *VCS_option_file*

Same as the -f option but allows you to specify a path to the file and the source files listed in the file do not have to be absolute pathnames.

-vcs *VCS options*

Specifies that the string enclosed within quotation marks is VCS options or Verilog files.

-define *define*

Sets the define option before reading the first Verilog file. The *define* argument might consist of a single value or a space-delimited list of values enclosed within braces.

-95 | -01 | -05

Specifies the IEEE standard to use for language interpretation. The default is -05, which specifies the IEEE 1364-2005. Use only the -95 and -01 options with the *-netlist* option. You can set the **hdlin_vrlg_std** variable to 1995, 2001, or 2005 to establish the Verilog language interpretation.

-extra_library_cells ***cell_list***

Specifies the list of design names(may be unused) to be elaborated. If called before set_top, the cells in cell_list will be elaborated during the subsequent set_top command even if they are not included in the top design hierarchy. This is in addition to the normal behavior of read_verilog. If the cells in cell_list already exist then they will be overwritten, which is the current behavior of read_verilog. If called after set_top in setup mode, or anytime in match or verify mode, the cells in cell_list (and only those designs) will be elaborated as part of the read_verilog command. If the library already exists, it will only be possible to add designs to it; that is, it will only be possible to elaborate designs that have not previously been elaborated. It will not be possible to overwrite any existing design. Attempting to overwrite an existing design will result in an error message.

file_list

Specifies one or more Verilog files to read.

DESCRIPTION

Use this command to read one or more Verilog files. Designs are read into design libraries, and Verilog library cell descriptions are read in as technology libraries. By default, the tool places designs into the default design library named *WORK*, and cell descriptions into the default technology library named *TECH_WORK*. The command, by default, detects if files are in compressed gzip format and reads them.

If a specified library does not exist, the tool creates it. If the library exists, it overwrites the existing designs or cell descriptions with the designs or cell descriptions that are read. Designs linked to any overwritten designs are unlinked.

If a Verilog file contains a structural netlist without the RTL source, use the *-netlist* option, which identifies the file as a Verilog structual netlist. This reduces the time required to read the file.

The following list describes how to use the **read_verilog** command.

- **Reading designs into a specific container** To read designs into a specific container, use the **read_verilog** command. Do not use the *-technology_library* option, instead use the *-r*, *-i*, or *-container* options. If the specified container does not exist, the tool creates the container and sets it as the current container. The syntax to use the command is:

```
read_verilog  -container c_name file
```

- **Reading designs in Verilog structural netlist format** To read designs in Verilog structural netlist format, use the for specific container, use the *-netlist* option. If the specified container does not exist, the tool reports an error. The syntax to use the command is:

```
read_verilog  -netlist
```

- **Reading unshared technology libraries** To read a technology library into a specific container, use the *-technology_library* option and the *-r*, *-i*, or *-container* options. If the specified container does not exist, the tool creates the container and sets it as the current container. The syntax to use the command is:

```
read_verilog  -technology_library -container c_name cells.v
```

- **Renaming a technology library** To rename a technology library, specify the *-libname*, or the *-work_library*, and *-technology_library* options along with the *-r*, *-i*, or *-container* options. By default, the command reads cell descriptions into the default technology library named *TECH_WORK*. The syntax to use the command is:

```
read_verilog -technology_library -r -libname my_tech_lib cells.v
```

- **Renaming a design library** To rename a design library, specify the *-libname* or *-work_library* option and do not use the *-technology_library* option. By default, the command reads designs into the default design library named *WORK*. The syntax to use the command is:

```
read_verilog  -libname my_design_lib file
```

- **VCS options** The **read_verilog** command supports some VCS options: *-v*, *-y*, *-f*, *+define*, *+libext* and *+incdir*. Some VCS options are irrelevant, therefore recognized but ignored: *-P* and all the other '+' options. The syntax of the command is:

```
read_verilog  -vcs "VCS options"
read_verilog  -f VCS_option_file
read_verilog  -F VCS_option_file
```

Special Behavior in the Library Verification Mode:

The library_verification command preprocesses all Verilog library cells into specified container. The container library names are handled internally by the tool. The only options that are valid in the library verification mode are *-r*, *-i*, and *-technology_library*.

```
fm_shell> read_verilog -r vlg_cells.v
Loading verilog file 'vlg_cells.v '
Preprocessing library 'vlg_cells.v' ...
Total cells found: 5
1
```

EXAMPLES

This example reads a Verilog design into the Formality environment. In this case, the *-container* option creates the container *ref*. The design files are read into a default design library named *WORK*, and the newly created container is established as the current container.

```
fm_shell> read_verilog -container ref p0.v
Loading verilog file '/u/formality/designs/p0.v'
No target library specified, default is WORK
Created container 'ref'
Current container set to 'ref'
1
fm_shell> report_design_libraries

Number of
Designs     Design Library
---------  --------------
    1        ref:/WORK
1
fm_shell>
```

This example reads a Verilog design into the current container. Here, the *-libname* option renames the design library to *my_lib*.

```
fm_shell> read_verilog -libname my_lib p0.v
Loading verilog file '/u/formality/designs/p0.v'
1
fm_shell> report_design_libraries -short
Number of
 Designs     Design Library
---------    --------------
    1        ref:/my_lib
1
fm_shell>
```

This example reads a technology library into container 'ref'.

```
fm_shell (setup)> read_verilog -technology_library -container ref lib.v
Loading verilog file '/u/formality/libraries/lib.v '
No target library specified, default is TECH_WORK
Created container 'ref'
Current container set to 'ref'
1
fm_shell (setup)> report_libraries -short

Number of
  Cells      Shared    Technology Library
---------    ------    ------------------
     83       Yes      GTECH

     83       Yes      i:/GTECH

     83       Yes      r:/GTECH

     83       Yes      ref:/GTECH
      1       No       ref:/TECH_WORK

1

This example reads a design and libraries using the -vcs option.
fm_shell (setup)> read_verilog -i ./netlist/design.v -vcs "-v newDFF.v
       -y ./libs/lib1 -y ./libs/lib2 +libext+.v"
Loading verilog file '/u/formality/netlist/design.v'
No target library specified, default is WORK
Current container set to 'i'
1
```

SEE ALSO

```
read_sverilog(2)
report_designs(2)
report_design_libraries(2)
report_libraries(2)
```

read_vhdl

Reads one or more VHDL files into the Formality environment.

SYNTAX

```
read_vhdl
   [ -container container_name | -r | -i ]
   [ -libname libname ]
   [ -work_library libname ]
   [ -technology_library ]
   [ -87 | -93 | -2008 ]
   file_names
```

Data Types

```
   container_name string
   libname string
   file_names string
```

ARGUMENTS

-container *container_name*

Specifies the container into which VHDL designs or cell descriptions are read.

-r

Reads VHDL designs or cell descriptions into the default reference container.

-i

Reads VHDL designs or cell descriptions into the default implementation container.

-libname *libname*

Specifies the design library or technology library into which VHDL designs or cell descriptions are read.

-work_library *libname*

Same behavior as the *-libname* option.

-technology_library

Identifies the file as a technology library. This option must be used with the *-r*, *-i*, or *-container* options.

-87 | -93 | -2008

Specifies the IEEE standard to use for language interpretation. The *hdlin_vhdl_std* variable defines the current default.

file_list

Specifies one or more VHDL files.

DESCRIPTION

This command reads one or more VHDL files into a container and sets the container as the current container. Designs are read into design libraries and library cell descriptions are read into technology libraries. If you do not specify the design or technology library, the command by default names the design library *WORK* and the default technology library *TECH_WORK*.

If the specified library name does not exist, the command creates a new library. If the specified library exists in the destination container, the command overwrites existing designs or cell descriptions. Designs linked to any overwritten designs become unlinked.

The following list shows how to use the command.

- Reading designs into a specific container To read designs into a specific container, use the *-r*, *-i*, or *-container* options. Do not specify the *-technology_library* option. If you use the *-container* option and the container does not exist, the command creates the container and establishes it as the current container. The syntax to use the command is:

  ```
  read_vhdl -container c_name file
  ```

- Reading unshared technology libraries To read a technology library into a specific container, use the *-technology_library* option and the *-r*, *-i*, or *-container* options. If the container does not exist, the command creates the container and establishes it as the current container. The syntax to use the command is:

  ```
  read_vhdl -technology_library -container c_name tech_file
  ```

- Renaming a technology library To rename a technology library, specify the *-libname*, or *-work_library*, and the *-technology_library* options along with the *-r*, *-i*, or *-container* option. By default, the command reads into the default TECH_WORK technology library. The syntax to use the command is:

  ```
  read_vhdl  -technology_library -r -libname my_tech_lib tech_file
  ```

- Renaming a design library To rename a design library, use the *-libname*, or *-work_library*, option and omit the *-technology_library* option. By default, the command reads designs into the default WORK design library. The syntax to use the command is:

  ```
  read_vhdl  -libname my_design_lib file
  ```

- Reading a design with VHDL-87 or VHDL-93 By default the command reads VHDL files as specified by the **hdlin_vhdl_std** variable. The default for this variable is *2008*. You can override the current variable setting by using the **read_vhdl** *-87* or *-93* command.

The **read_vhdl** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example reads a VHDL design into the Formality environment. The *-container* option creates a container names *ref*. The tool reads the design files into the WORK design library and establishes the new container as the current container.

```
fm_shell> read_vhdl -container ref des_1.vhd
Loading vhdl file '/u/designs/des_1.vhd'
No target library specified, default is WORK
1
fm_shell>
```

The following example reads a VHDL design into the current container. The *-libname* option renames the design library *my_lib*.

```
fm_shell> read_vhdl -libname my_lib des_1.vhd
Loading vhdl file '/u/designs/des_1.vhd'
1
fm_shell>
```

The following example reads a shared technology library into all containers. The technology library is automatically loaded into subsequent new containers.

```
fm_shell> read_vhdl -technology_library des_lib.vhd
Loading vhdl file '/u/designs/des_lib.vhd'
No target library specified, default is TECH_WORK
1
fm_shell>
```

SEE ALSO

```
report_designs(2)
report_design_libraries(2)
report_libraries(2)
hdlin_vhdl_std(3)
```

record_edits

Records edit commands.

SYNTAX

```
record_edits
   [ -on | -off ]
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-on

Turns recording of edit commands on.

-off

Turns recording of edits commands off.

DESCRIPTION

This command is used to control the recording of edit commands (**create_net**, **disconnect_net**, ...). Recording is on by default when Formality starts.

Called without an argument, the command turns recording on.

Use the **write_edits** command to write the recorded edit commands to a file.

SEE ALSO

```
create_net(2)
disconnect_net(2)
report_edits(2)
write_edits(2)
```

redirect

Redirects the output of a command to a file.

SYNTAX

```
string redirect
   [-append]
   [-tee]
   [-file | -variable]
   target
   {command_string}
```

Data Types

```
target
command_string
```

ARGUMENTS

-append

Appends the output to the specified target file.

-tee

Like the unix command of the same name, sends output to the current output channel as well as to the *target*.

-file

Indicates that *target* is a file name, and redirection is to that file. This is the default. It is exclusive of *-variable*.

-variable

Indicates that *target* is a variable name, and redirection is to that Tcl variable. It is exclusive of *-file*.

target

Indicates the target of the output redirection. If redirecting to a file, this is the file name. If redirecting to a Tcl variable, this is the variable name.

command_string

The command to execute. Intermediate output from this command, as well as the result of the command, will be redirected to *target*. The *command_string* should be rigidly quoted with curly braces.

DESCRIPTION

This command performs the same function as the traditional unix-style redirection operators > and >>. The *command_string* must be rigidly quoted (that is, enclosed in curly braces) in order for the operation to succeed. It must not be constructed as a nested command.

Output is redirected to a file by default. Output can be redirected to a Tcl variable by using the *-variable* option.

Output can be channeled to the current output device as well as the redirect target by using the *-tee* option. See the examples section for an example.

The result of a **redirect** command which does not generate a Tcl error is the empty string. Screen output occurs only if errors occurred during execution of the *command_string* (other than opening the redirect file). When errors occur, a summary message is output. See the examples.

Although the result of a successful **redirect** command is the empty string, it is still possible to get and use the result of the command that you redirected. Construct a **set** command in which you set a variable to the result of your command. Then, redirect the **set** command. The variable holds the result of your command. See the examples.

The **redirect** command is much more flexible than traditional unix redirection operators. With **redirect**, you can redirect multiple commands or an entire script. See the examples for an example of how to construct such a command.

Note that the builtin Tcl command **puts** does not respond to output redirection of any kind. Use the builtin **echo** command instead.

EXAMPLES

In the following example, the output of the plus procedure is redirected. The echoed string and the result of the plus operation is in the output file. Notice that the result was not echoed to the screen.

```
prompt> proc plus {a b} {echo "In plus" ; return [expr $a + $b]}
prompt> redirect p.out {plus 12 13}
prompt> exec cat p.out
In plus
```

```
25
```

In this example, a typo in the command created an error condition. The error message indicates that you can use **error_info** to trace the error, but you should first check the output file.

```
prompt> redirect p.out {plus2 12 13}
Error: Errors detected during redirect
        Use error_info for more info. (CMD-013)
prompt> exec cat p.out
Error: unknown command 'plus2' (CMD-005)
```

In this example, we explore the usage of results from redirected commands. Since the result of **redirect** for a command which does not generate a Tcl error is the empty string, use the **set** command to trap the result of the command. For example, assume that there is a command to read a file which has a result of "1" if it succeeds, and "0" if it fails. If you redirect only the command, there is no way to know if it succeeded.

```
redirect p.out { read_a_file "a.txt" }
# Now what?  How can I redirect and use the result?
```

But if you set a variable to the result, then it is possible to use that result in a conditional expression, etc.

```
redirect p.out { set rres [read_a_file "a.txt"] }
if { $rres == 1 } {
  echo "Read ok!"
}
```

The **redirect** command is not limited to redirection of a single command. You can redirect entire blocks of a script with a single **redirect** command. This simple example with **echo** demonstrates this feature:

```
prompt> redirect p.out {
?         echo -n "Hello "
?         echo "world"
?       }
prompt> exec cat p.out
Hello world
prompt>
```

The **redirect** command allows you to tee output to the previous output device and also to redirect output to a variable. This simple example with **echo** demonstrates these features:

```
prompt> set y "This is "
This is
prompt> redirect -tee x.out {
          echo XXX
          redirect -variable y -append {
            echo YYY
            redirect -tee -variable z {
              echo ZZZ
            }
          }
        }
XXX
prompt> exec cat x.out
XXX
prompt> echo $y
This is YYY
ZZZ

prompt> echo $z
ZZZ
```

SEE ALSO

```
echo(2)
error_info(2)
set(2)
```

remove_black_box

Removes user-defined black boxes.

SYNTAX

```
remove_black_box
   [ object_list ]
   [ -attribute attribute_name ]
   -all
```

Data Types

```
object_list string
attribute_name string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

object_list

Removes black boxes from the specified designs or cells.

-all

Removes black boxes from all designs and cells in the Formality environment.

-attribute *attribute_name*

Removes black boxes from all designs that have the specified attribute.

DESCRIPTION

This command removes black boxes that are set by using the **set_black_box** command.

You can remove black boxes from all designs or cells from individual designs and cells. To remove black boxes from all designs and cells, use the *-all* option or do not specify any arguments. For specific designs, use one or more object lists.

The **remove_black_box** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The first example removes a black box from the design named *lower*. The second example removes a black box from the cell named *inst1* in the top module *top*.

```
fm_shell (setup)> remove_black_box rtl:/WORK/lower
Removed user-defined black box on 'rtl:/WORK/lower'
1
fm_shell (setup)> remove_black_box rtl:/WORK/top/inst1
Removed user-defined black box on 'rtl:/WORK/top/inst1'
1
fm_shell (setup)>
```

The following example removes all user-defined black boxes from all designs and cells in the Formality environment.

```
fm_shell (setup)> remove_black_box -all
Removed all user-defined black boxes
1
fm_shell (setup)>
```

SEE ALSO

```
report_black_box(2)
set_black_box(2)
```

remove_cell

Removes cells.

SYNTAX

```
status remove_cell
   cell_list
   -all
```

Data Types

```
cell_list list
```

RETURN VALUE

The **remove_cell** command returns a status of 1 if it was successful and 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

cell_list

Specifies the names of cells to remove. You can use either object IDs or instance-based paths.

Cells are removed from all instances of a design. If you want to only affect a particular instance-based path, you must uniquify that instance of the design first.

You must specify either the *cell_list* or *-all*.

-all

Removes all cells in the current design or instance.

You must specify either the *cell_list* or *-all*.

DESCRIPTION

This command removes the cells that are specified using the *cell_list* argument, or all cells in the current design or instance if the *-all* option is used.

This command disconnects nets that are connected to pins on the cells that are removed.

You can create cells using the **create_cell** and **create_primitive** commands.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example removes two cells *box1* and *box2* in the design r:/WORK/mid.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> remove_cell {box1 box2}
```

The following example removes cell *box3* from all instances of the design of r:/WORK/top/m1/b1.

```
fm_shell (setup)> current_instance r:/WORK/top/m1/b1
fm_shell (setup)> remove_cell box3
```

SEE ALSO

```
create_cell(2)
create_primitive(2)
current_design(2)
current_instance(2)
edit_design(2)
```

remove_cell_type

Removes the cell type that was set on a technology library cell or collection of cells

SYNTAX

```
remove_cell_type
   [ design_list ]
   -all
```

Data Types

```
design_list string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

object_list

Specifies a tech-lib cell whose cell-type value needs to be removed. It is the string name of exactly one tech-lib cell or a collection of one or more cells.

-all

Removes the cell-type from all user specified cell-types in tech-lib cells

DESCRIPTION

Removes the cell type associated with the technology library cell or a collection of one or more cells.

EXAMPLES

The first example removes the cell-type lssd from the design \fr:/LIBNAME/Ilower.

```
fm_shell (setup)> remove_cell_type r:/LIBNAME/cellname
r:/LIBNAME/cellname: Removed as LSSD
1
fm_shell (setup)>
```

The first example removes the cell-type retention from the design \fr:/LIBNAME/Ilower.

```
fm_shell (setup)> remove_cell_type r:/LIBNAME/cellname
r:/LIBNAME/cellname: Removed as RETENTION
1
fm_shell (setup)>
```

The following example removes all user-defined cell-types from the tech-lib cells .

```
fm_shell (setup)> remove_cell_type -all
1
fm_shell (setup)>
```

SEE ALSO

```
report_cell_type(2)
set_cell_type(2)
```

remove_clock

Removes clocks that are created by using the **set_clock** command.

SYNTAX

```
remove_clock
   netID | -all
```

Data Types

```
   netID string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

netID

Specifies a list of clocks to remove. If you specify a name consisting of a regular expression that resolves to more than one net, the operation is applied to the matching nets.

-all

Removes all clocks from the current design.

DESCRIPTION

This command removes user-specified clocks that are created by using the **set_clock** command.

Specify either a clock name or the *-all* option.

EXAMPLES

```
fm_shell (setup)> remove_clock HDL:/WORK/CORE/SCLK
Removed clock from 'HDL:/WORK/CORE/SCLK'
1
fm_shell (setup)> remove_clock -all
Removed all user-defined clocks
1
```

SEE ALSO

```
report_clocks(2)
set_clock(2)
```

remove_compare_rules

Removes all user-defined compare rules.

SYNTAX

```
remove_compare_rules
   [ designID ]
```

Data Types

```
designID string
```

ARGUMENTS

designID

Specifies the design from which to remove the user-defined compare rules.

DESCRIPTION

Use this command to remove all previously defined name matching rules, or compare rules, from a specified design. For more information about compare rules, see the *Formality User Guide*.

If you do not specify a value for the *designID* argument, the command removes compare rules from the current design. If you specify a design ID that has no compare rules, the command issues a warning message.

The **remove_compare_rules** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

This example removes all compare rules in the ff.async.reset design, in the current container.

```
fm_shell> remove_compare_rules ff.async.reset
1
fm_shell>
```

This example removes all compare rules in the current design.

```
fm_shell> remove_compare_rules
1
fm_shell>
```

SEE ALSO

```
report_compare_rules(2)
set_compare_rule(2)
```

remove_constant

Removes the user-defined constants specified by the **set_constants** command.

SYNTAX

```
remove_constant
   -all | objectID
   [-type type]
```

Data Types

```
objectID string
type string
```

Enabled Shell Modes

Setup

ARGUMENTS

-all

Removes all user-defined constants from designs in the Formality environment.

objectID

Specifies the design object constant you want to remove. If you specify a name consisting of a regular expression that resolves to more than one object, the operation is applied to all matching objects, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected. The order of precedence is pin, port, net, cell.

-type *type*

Specifies the object type of the specified object ID. Use this option if the name of the specified design object is associated with more than one object type within the same design. Specify one of the following values for the *type* argument:

- *pin* specifies pin type
- *port* specifies port type

- *net* specifies net type
- *cell* specifies cell type

DESCRIPTION

This command removes the specified constants that are specified by the **set_constant** command.
To remove a constant from a specific design object, specify the *objectID*. If the design object is located in the current design and container, you do not have to specify the container and design information.
When design objects of different types have the same name within a design, use the **-type** option.
Use the **report_constants** command to report the ports and nets that are defined as constants.

The **remove_constant** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example removes the constant previously set at port CC. The design object is located in the implementation container, the WORK library, and the CORE design.

```
fm_shell (setup)> remove_constant impl:/WORK/CORE/CC

1

fm_shell (setup)>
```

The following example removes the constant previously set at net B7. The design also contains a port with the same name, so you must specify the **-type** option. The net is located in the current design.

```
fm_shell (setup)> remove_constant -type net B7

1

fm_shell (setup)>
```

SEE ALSO

```
report_constants(2)
set_constant(2)
```

remove_constraint

Removes external constraints from the control points of a design.

SYNTAX

```
status remove_constraint
   constraint_name
```

Data Types

```
constraint_name string
```

ARGUMENTS

constraint_name

Specifies the name of the constraint to remove set by the **set_constraint** command.

DESCRIPTION

Use this command to remove a specified constraint from the Formality session. Use the **report_constraint** command to report a list of constraints.

EXAMPLES

This example removes a constraint.

```
fm_shell> set_constraint 0hot {IN1 IN2 IN3 IN4} FM_CONSTRAINT_0
fm_shell> remove_constraint FM_CONSTRAINT_0
1
```

SEE ALSO

```
create_constraint_type(2)
remove_constraint_type(2)
report_constraint(2)
report_constraint_type(2)
set_constraint(2)
```

remove_constraint_type

Removes the specified external constraint type that is created by using the **create_constraint_type** command.

SYNTAX

```
remove_constraint_type
   type_name
```

Data Types

```
type_name string
```

ARGUMENTS

type_name

Specifies the type of user-specified constraint to remove.

DESCRIPTION

Use this command to remove user-defined external constraint types created by using the **create_constraint_type** command. Constraints that are of the removed type are also removed.

Do not use this command to remove predefined constraint types. See the **set_constraint** command man page for a list of predefined types.

The **remove_constraint** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

This example removes a constraint type that is used by an existing constraint.

```
fm_shell> set_constraint parity {IN1 IN2 IN3} -map \
{A=IN1 B=IN2 C=IN3} $ref FM_CONSTRAINT_0

fm_shell> remove_constraint_type parity
Information: Constraint 'FM_CONSTRAINT_0' uses type
'parity' and is being removed. (FM-087)
1
```

SEE ALSO

```
create_constraint_type(2)
remove_constraint(2)
report_constraint(2)
report_constraint_type(2)
set_constraint(2)
```

remove_container

Removes the specified container(s) from the Formality environment.

SYNTAX

```
remove_container
   container_list | -all
```

Data Types

```
container_list string
```

ARGUMENTS

container_list

Specifies the containers to remove. You can specify one or more containers to remove.

-all

Removes all containers in the current session and their contents.

DESCRIPTION

Use this command to remove a specific or all containers.

This command removes all design libraries, technology libraries, and the container. Unsaved information in the container, such as user-defined constants, is also removed. To save the contents of a container use the **write_container** command.

The **remove_container** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example removes all the containers in the Formality environment:

```
fm_shell> remove_container -all
Removed container 'impl'
Current design is no longer set
Implementation design is no longer set
Removed container 'ref'
Reference design is no longer set
1
fm_shell> report_containers
containers:     None
1
fm_shell>
```

SEE ALSO

```
report_containers(2)
report_design_libraries(2)
report_designs(2)
report_hierarchy(2)
report_libraries(2)
```

remove_cutpoint

Removes cutpoints that are specified by the **set_cutpoint** command or created by the **create_cutpoint_blackbox** command.

SYNTAX

```
remove_cutpoint
   objectID | -all
   [-type type]
```

Data Types

```
objectID string
type string
```

Enabled Shell Modes

Setup

ARGUMENTS

objectID

Specifies a list of cutpoints to remove. If you specify a name consisting of a regular expression that resolves to more than one object, the operation is applied to all the matching objects. If the specified name resolves to multiple objects (with identical names and unspecified object type), only one of these objects is affected. The precedence is pin then net.

-all

Removes all the cutpoints from the current design.

-type *type*

Specifies the object type. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following values for the *ID_type* argument:

- *pin* to specify the pin type

- *net* to specify the net type
- *cell* to specify the cell type

DESCRIPTION

This command removes user-specified cutpoints that are specified by using the **set_cutpoint** command or created by using the **create_cutpoint_blackbox** command.

Specify either a specific cutpoint with the objectID argument or use the **-all** option to remove all cutpoints in the current design.

EXAMPLES

```
fm_shell (setup)> remove_cutpoint HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[9]
Removed cutpoint from 'HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[9]'
1
fm_shell (setup)> remove_cutpoint -all
Removed all user-defined cutpoints
1
```

SEE ALSO

```
set_cutpoint(2)
report_cutpoints(2)
create_cutpoint_blackbox(2)
```

remove_design

Removes the specified designs from the Formality environment.

SYNTAX

```
remove_design
   [-hierarchy]
   [-shared_lib]
   designID_list
```

Data Types

```
designID_list string
```

Enabled Shell Modes

Setup

ARGUMENTS

-hierarchy

Removes all files recursively from hierarchical designs.

-shared_lib

Removes designs in the technology libraries.

designID_list

Removes one or more designs from the current container.

DESCRIPTION

This command removes designs from the Formality environment. By default, the tool removes the design from the current container unless you specify a container name as part of the *designID* argument.

You can recursively remove all files in a design's hierarchy by using the *-hierarchy* option. However, this option does not remove designs in the technology libraries.

NOTE: After you successfully run the **set_top** command on a container, its design can no longer be removed. Use the **set_black_box** command instead.

The **remove_design** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example removes the UPC design from the container named "ref" but does not remove the hierarchical files in it.

```
fm_shell (setup)> remove_design /WORK/UPC
Removed design '/WORK/UPC' from container 'ref'
1
fm_shell (setup)>
```

The following example removes the UPC design as in the preceding example. However, this command uses the *-hierarchy* option, which removes the UPC_DW01_add_12_0 design as well.

```
fm_shell (setup)> remove_design -hierarchy /WORK/UPC
Removed design '/WORK/UPC' from container 'ref'
Removed design '/WORK/UPC_DW01_add_12_0' from
container 'ref'
1
fm_shell (setup)>
```

SEE ALSO

```
remove_black_box(2)
report_black_boxes(2)
report_designs(2)
set_black_box(2)
```

remove_design_library

Removes one or more design libraries from the Formality environment.

SYNTAX

```
remove_design_library
   design_libraryID_list | -all
```

Data Types

```
   design_libraryID_list string
```

ARGUMENTS

design_libraryID_list

Removes one or more design libraries from the current container.

-all

Removes all design libraries from the current container.

DESCRIPTION

This command removes design libraries from the current container, unless you specify a container name as part of the *designID* argument.

You can remove all design libraries from the current container by using the **-all** option.

NOTE: Remove design libraries from containers before using the **set_top** command. After you successfully run the **set_top** command on a container, its design can no longer be removed.

The **remove_design_library** command returns one of the following:

- 0 to indicate failure

- 1 to indicate success

EXAMPLES

The following example removes the WORK design library from the default implementation container.

```
fm_shell> remove_design_library WORK
Removed design library 'WORK' from container 'impl'
1
fm_shell>
```

SEE ALSO

```
remove_design(2)
report_designs(2)
```

remove_dont_cut

Removes the dont-cutpoints that are specified using the **set_dont_cutpoint** command.

SYNTAX

```
remove_dont_cut
   objectID | -all
   [-type type]
```

Data Types

```
objectID string
type string
```

Enabled Shell Modes

Setup

ARGUMENTS

objectID

Specifies a list of dont-cutpoints to remove. If you specify a name consisting of a regular expression that resolves to more than one object, the operation is applied to all the matching objects. If the specified name resolves to multiple objects with identical names and unspecified object type, only one of these objects is affected. The precedence is pin then net.

-all

Removes all the dont-cutpoints from the current design.

-type *type*

Specifies the object type. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following object types:

- *pin* to specify the pin type
- *net* to specify the net type

- *cell* to specify the cell type

DESCRIPTION

This command removes the dont-cutpoints that are specified using the **set_dont_cut** command.

Specify either a specific cutpoint using the **objectID** option or use the **-all** option to remove all cutpoints in the current design.

EXAMPLES

```
fm_shell (setup)> remove_dont_cut $ref/block/IN
Removed dont_cut from 'r:/WORK/top/block/IN'
1
fm_shell (setup)> remove_dont_cut -all
Removed all user defined dont_cutpoints
1
```

SEE ALSO

```
set_dont_cut(2)
report_dont_cuts(2)
```

remove_dont_match_points

Removes a list of user-specified dont-match points.

SYNTAX

```
integer remove_dont_match_points
   [-type ID_type]
   objectID_list
   [-all]
```

Data Types

```
ID_type string
objectID_list string
```

ARGUMENTS

-type *ID_type*

Identifies the object type of the specified object. Use this option when the name of the specified design object is associated with more than one object type within the same design. Specify one of the following object types.

- **pin** - Specifies the pin type.
- **port** - Specifies the port type.
- **net** - Specifies the net type.
- **cell**- Specifies the cell type.

objectID_list

Removes the dont_match attribute from the specified design objects. If the specified regular expression resolves to more than one object, the operation is applied to all of the objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the following precedence, pin, port, net, cell.

-all

Removes the user-specified dont_match attributes from objects.

DESCRIPTION

This command removes the user-specified dont_match attributes from specified objects. To enable matching of all user-specified dont-match points, use the **-all** option.

EXAMPLES

The following example removes a user-defined dont-match point. The port *A_T33* is in the default implementation container, the *WORK* library, and the *CORE* design.

```
prompt> remove_dont_match_points \
impl:/WORK/CORE/A_T33
```

The following example is similar to the preceding example except that the **-type** option differentiates the design object from other types having the same name.

```
prompt> remove_dont_match_points \
-type cell \
impl:/WORK/CORE/A_T33
```

SEE ALSO

```
report_dont_match_points(2)
report_matched_points(2)
report_unmatched_points(2)
set_dont_match_points(2)
```

remove_dont_verify_points

Removes a specified list of user-defined dont-verify points.

SYNTAX

```
integer remove_dont_verify_points
   [-type ID_type]
   objectID_list
   [-all]
   [-directly_undriven_output]
```

Data Types

ID_type string *objectID_list* string

ENABLED SHELL MODES

Setup

ARGUMENTS

-type *ID_type*

Identifies the object type of the specified object. Use this option when the name of the specified design object is associated with more than one object type within the same design. Specify one of the following object types:

- **pin** - Specifies the pin type.
- **port** - Specifies the port type.
- **net** - Specifies the net type.
- **cell**- Specifies the cell type.

objectID_list

Specifies one or more design objects to re-enable. If you specify a name consisting of a regular expression that resolves to more than one object, the operation applies to all of the matching objects. However, if the specified name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the precedence of pin, port, net, and cell.

-all

Removes all user-defined dont-verify points from the current design.

-directly_undriven_output

Specifies all top-level reference design output ports that either do not have connected nets, or that have connected nets but do not have driving pins.

DESCRIPTION

This command removes the dont_verify attribute from the specified compare points. To remove the attribute from all user-specified don't-verify points, use the **-all** option.

Don't-verify points are not removed from the designs until you issue a subsequent **match** or **verify** command. To report the dont-verify points using the following commands:

```
report_matched_points -point_type dont_verify

report_unmatched_points -point_type dont_verify
```

EXAMPLES

The following example removes the dont_verify attributes. Port *A_T33* is in the default implementation container, the *WORK* library, and the *CORE* design.

```
prompt> remove_dont_verify_points \
impl:/WORK/CORE/A_T33
```

The following example is similar to the preceding example except that the **-type** option differentiates the design object from other types having the same name.

```
prompt> remove_dont_verify_points \
-type cell \
impl:/WORK/CORE/A_T33
```

SEE ALSO

```
report_dont_verify_points(2)
report_matched_points(2)
report_unmatched_points(2)
set_dont_verify_points(2)
```

remove_dp_int_round

Removes any rounding information for multipliers.

SYNTAX

```
remove_dp_int_round
   objectID
```

Data Types

```
objectID string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

objectID

Removes any rounding information for the multiplier.

DESCRIPTION

Use this command to remove the internal and external rounding modifications applied to a multiplier.

EXAMPLES

```
     fm_shell (guide)> **remove_dp_int_round** { mult_12* mult_34* }
     1
```

SEE ALSO

```
   set_dp_int_round(2)
   report_dp_int_round(2)
```

remove_factor_point

Removes the design object as a factoring variable specified by the **set_factor_point** command.

SYNTAX

```
remove_factor_point
   objectID | -all
   [-type type]
```

Data Types

```
objectID string
type string
```

Enabled Shell Modes

- Setup
- Match
- Verify

ARGUMENTS

objectID

Specifies a list of factoring variables to remove. If you specify a name consisting of a regular expression that resolves to more than one object, the operation is applied to all the matching objects. If the specified name resolves to multiple objects with identical names (and you do not specify the object type), only one of these objects is affected. In which case, the precedence is pin, port, net, and cell.

-all

Removes all factoring variables from the current design.

-type *type*

Specifies the object type. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following values for the *objectID_type* argument:

- *port* to specify port type
- *pin* to specify pin type
- *net* to specify net type
- *cell* to specify cell type

DESCRIPTION

This command removes the given objects as factoring variables, which are specified by using the **set_factor_point** command. Specify either a list of objects with valid factoring variable names with the objectID or use the **-all** option. Use wildcard character (*) to specify groups of objects.

EXAMPLES

```
fm_shell (setup)> remove_factor_point $ref/M1/U1/state1_reg
Removed factoring variable from 'r:/WORK/top/M1/U1/state1_reg'
1
fm_shell (setup)> remove_factor_point -all
Removed all user-defined factoring variables
1
```

SEE ALSO

```
set_factor_point(2)
report_factor_points(2)
```

remove_from_collection

Removes objects from a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection remove_from_collection
   [-intersect]
   collection1
   object_spec
```

Data Types

```
collection1          collection
object_spec          list
```

ARGUMENTS

-intersect

Removes objects from collection1 not found in object_spec. Without this option, removes objects from collection1 that are found in object_spec.

collection1

Specifies the base collection to be copied to the result collection. Objects matching *object_spec* are removed from the result collection.

object_spec

Specifies a list of named objects or collections to remove. The object class of each element in this list must be the same as in the base collection. If the name matches an existing collection, the collection is used. Otherwise, the objects are searched for in the database using the object class of the base collection.

DESCRIPTION

The **remove_from_collection** command removes elements from a collection, creating a new collection.

If the base collection is homogeneous, any element of the *object_spec* that is not a collection is searched for in the database using the object class of the base collection. If the base collection is heterogeneous, any element of the *object_spec* that is not a collection is ignored.

If the *-intersect* option is not specified, which is the default mode, and if nothing matches the *object_spec*, the resulting collection is a copy of the base collection. If everything in the *collection1* option matches the *object_spec*, the result is the empty collection. With the *-intersect* option the results are reversed.

For background on collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example gets all input ports except "CLOCK".

```
prompt> set cPorts [remove_from_collection [all_inputs] CLOCK]
{"in1", "in2"}
```

SEE ALSO

```
add_to_collection(2)
collections(2)
```

remove_guidance

Removes the current setup data specified by the **set_svf** command

SYNTAX

```
remove_guidance
```

Enabled Shell Modes

- Setup
- Guide

DESCRIPTION

This command removes the stored setup data, which is specified by the **set_svf** command.

EXAMPLES

```
fm_shell (setup)> remove_guidance
SVF is not set.
0
fm_shell (setup)> set_svf top.svf
SVF set to 'top.svf'.
1
fm_shell (setup)> remove_guidance
Removed SVF from 'top.svf'.
1
fm_shell (setup)>
```

SEE ALSO

```
set_svf(2)
report_guidance(2)
```

remove_init_toggle_assumption

Removes the initial toggle assumption on a controlling object(s).

SYNTAX

```
remove_init_toggle_assumption
   [-type]
   [-all | object_list]
```

Data Types

object_list string

Enabled Shell Modes

Setup

ARGUMENTS

-type

Optional switch to specify the type of the object(s). The types are port, net or cell.

object_list

Specifies the object(s) whose initial toggle assumptions need to be removed.

-all

Removes all the initial toggle assumptions.

DESCRIPTION

Removes the initial toggle assumption on a controlling object(s) set by the set_init_toggle_assumption command.

EXAMPLES

The following example removes the initial toggle assumption from *i:/WORK/bit_slice/hclk*.

```
fm_shell (setup)> remove_init_toggle_assumption i:/WORK/bit_slice/hclk
Removing the initial toggle assumption of i:/WORK/bit_slice/hclk
1
```

The following example removes all the initial toggle assumptions..

```
fm_shell (setup)> remove_init_toggle_assumption -all
Removing the initial toggle assumption of i:/WORK/bit_slice/hclk
Removing the initial toggle assumption of i:/WORK/bit_slice/lclk
1
```

SEE ALSO

```
set_init_toggle_assumption(2)
report_init_toggle_assumption(2)
```

remove_input_value_range

Removes a specified or all input value ranges that are set by using the **set_input_value_range** command.

SYNTAX

```
remove_input_value_range
   objectID | -all
```

Data Types

```
objectID string
```

Enabled Shell Modes

Setup

ARGUMENTS

objectID

Removes input value range for the specified object ID.

-all

Removes all input value ranges.

DESCRIPTION

This command removes the input value ranges that are set by the **set_input_value_range** command. All primary inputs default to binary (0 or 1) value range.

The **remove_input_value_range** command returns one of the following:

- 0 to indicate failure

- 1 to indicate success

EXAMPLES

The following example removes the input value range for r:/WORK/top/PI1.

```
fm_shell (setup)> remove_input_value_range r:/WORK/top/PI1
Removed user-defined input range from 'r:/WORK/top/PI1'
1
```

The following example removes all the input value ranges.

```
fm_shell (setup)> remove_input_value_range -all
Removed all user-defined input ranges
1
fm_shell (setup)>
```

SEE ALSO

```
set_input_value_range(2)
report_input_value_range(2)
write_hierarchical_verification_script(2)
```

remove_inv_push

Disables the move of inversions across the register boundaries.

SYNTAX

```
remove_inv_push
   [-shared_lib]
   objectID_list | -all
```

Data Types

objectID_list

Enabled Shell Modes

Setup

ARGUMENTS

-shared_lib

Disables the move of inversions across the register boundaries from the shared technology libraries.

objectID_list

Specifies the objects where the inversion across the register boundaries are disabled.

-all

Disables all inversions across register boundaries.

DESCRIPTION

This command disables the move of an inversion across the register boundary from an object. You can use this command only on:

- A sequential primitive in a design
- An instance referencing a technology library
- A design in a technology library
- A top-level, or primary output port
- An input pin on a black box

The **remove_inv_push** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example specifies the removal of the inversion across the reg_A, register boundary.

```
fm_shell (setup)> remove_inv_push ref:/WORK/CORE/reg_A
1
```

SEE ALSO

```
report_inv_push(2)
set_inv_push(2)
```

remove_inversion

Removes user-specified inversions.

SYNTAX

```
status remove_inversion
   objectID | -all
```

Data Types

```
objectID string
```

ENABLED SHELL MODES

Setup

ARGUMENTS

objectID

Removes the specified inversion objects.

-all

Removes all the user-specified inversion objects from the current design.

DESCRIPTION

This command removes the specified inversion object that is inserted using the **insert_inversion** command.

To remove an inversion object from a specific design object, specify the *objectID* of the inversion object. To remove all user-inserted inversion from all workspaces, specify the **-all** option.

The **remove_inversion** command is available from the GUI through the "Setup" menu of the logic cone schematics. You must select an inversion object before "Remove inversion" is enabled in the "Setup menu". The GUI queues the command, because the **remove_inversion** command must be executed in the setup mode. NOTE that the **remove_inversion\f command is made on the actual design, so inversion is removed from all instantiations of the design. The tool reports a warning from the GUI if inversions are removed in more places than just the selection. No warning is reported from the command line.**

To report the inversion objects that are inserted using the **insert_inversion** command, use the **report_inversion** command.

EXAMPLES

The following example shows how to remove inversion objects.

```
fm_shell (setup)> remove_inversion i:/WORK/top/out2_FM_INV
1
```

The following example removes all the inversion previously created with **insert_inversion** commands.

```
fm_shell (setup)> remove_inversion -all
1
```

SEE ALSO

```
report_inversion(2)
insert_inversion(2)
```

remove_library

Removes the specified technology libraries from the Formality environment.

SYNTAX

```
remove_library
   libraryID_list | -all
```

Data Types

```
libraryID_list string
```

ARGUMENTS

libraryID_list

Specifies the libraries to remove. You can specify the name of the library, or the library ID including the name of the container.

-all

Removes the technology libraries that are currently loaded in all containers.

DESCRIPTION

This command removes the technology libraries from the Formality environment. You can either remove all libraries, including the GTECH libraries, or specific libraries. Libraries that are removed are not loaded (with the exception of GTECH) into subsequently created containers.

You can remove specific libraries by listing the library names. Do not specify the container or use the slash character as you would when specifying a library ID argument.

To remove unshared technology libraries, specify the *libraryID* argument.

Removing a shared technology library from a specific container does not prevent the tool from automatically loading that library into subsequently created containers.

The **remove_library** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example removes the shared technology library named *my_shared_lib*. The report shows all the currently loaded libraries. After the tool removes *my_shared_lib*, it is not loaded into the newly created containers.

```
fm_shell> report_libraries -short

Number of
 Cells      Shared    Technology Library
---------   ------    ------------------
    1        Yes      my_shared_lib
    79       Yes      gtech
    1        Yes      ref:/my_shared_lib
    79       Yes      ref:/gtech
1

fm_shell> remove_library my_shared_lib
Removed shared technology library 'my_shared_lib'
1

fm_shell> report_libraries -short
Number of
Cells      Shared    Technology Library
---------  ------    ------------------
    79      Yes      gtech
    79      Yes      ref:/gtech
1
fm_shell>
```

SEE ALSO

```
report_libraries(2)
```

remove_license

Removes one or more licensed Formality features.

SYNTAX

```
status remove_license
   feature_list
```

Data Types

```
feature_list list
```

ARGUMENTS

feature_list

Specifies the list of Formality features to be removed. The *feature_list* argument might consist of a single value or a space-delimited list of values enclosed within braces ({}).

By looking at your key file, you can determine all of the features licensed at your site.

RETURN VALUE

The **remove_license** command returns a status of 1 if it was successful and 0 if it failed.

DESCRIPTION

This command removes the specified Formality license features from the features you are currently using.

The **list_licenses** command provides a list of the features that you are currently using.

EXAMPLES

The following example removes the Formality-Ultra license:

```
fm_shell (setup)> remove_license Formality-Ultra
```

SEE ALSO

```
check_license(2)
license_users(2)
list_licenses(2)
get_license(2)
```

remove_mismatch_message_filter

Removes warning or suppress filter on one or more simulation-synthesis mismatch messages.

SYNTAX

```
status remove_mismatch_message_filter
   [-warn | -suppress]
   [-all]
   [-signal SignalName]
   [-block HierarchicalBlockName]
   [-file FileName]
   [-line LineNumber]
   [MismatchMessageIDList]
```

Data Types

```
   SignalName                 string
   HierarchicalBlockName      string
   FileName                   string
   LineNumber                 integer
   MismatchMessageIDList      list
```

ARGUMENTS

-warn

Removes mismatch filter set with **-warn** option. This option can not be combined with **-suppress**.

-suppress

Removes mismatch message filter set with **-suppress** option. This option can not be combined with **-warn**.

-all

This option removes all filters. This option can be combined with **-warn** or **-suppress**. The option **-all** alone will remove all the filters set previously. It removes all warning filters if specified with **-warn** option and removes all suppress filters if specified with **-suppress** option.

-signal SignalName

Removes mismatch message filter based on the Signal or Variable Name. The SignalName can accept a string value in Tcl glob style pattern that should exactly match the **set_mismatch_message_filter** input. This is an optional option.

`-block HierarchicalBlockName`

Removes mismatch message filter based on the hierarchical block name. The Hierarchical block name can accept a string value in Tcl glob style pattern that should exactly match the **set_mismatch_message_filter** input. The block name can be module or entity-architecture name, always or process block name, generate block name, function name and procedure name. The expected format of hierarchical block name for always or process block and generate block are as follows:

<ModuleName>[/<BlockName>]*

The expected format of hierarchical block name for functions and procedures that defined inside a module or package is as follows:

<ModuleOrPackageName>/<FunctionOrProcedureName>

This is an optional option.

`-file FileName`

Removes mismatch message filter based on a file name. The file name can be a leaf level file name like */test.v or full file path. This can accept values which are in Tcl glob-style pattern matching form that exactly matches **set_mismatch_message_filter** input. This is an optional option.

`-line LineNumber`

Removes mismatch message filter for a mismatch message that occurs at given line number of an RTL file. This option requires **-file** option. It cannot be used with **-signal** or **-block** options. This is an optional option

`MismatchMessageIDList`

Removes mismatch message filter for the list of mismatch message Ids specified. This is an optional option. Below are the list of of simulation mismatch error codes that this option accepts:

FMR_VHDL-274 FMR_VHDL-1002 FMR_VHDL-1004 FMR_VHDL-1014 FMR_VHDL-1025 FMR_VHDL-1027 FMR_VHDL-1036 FMR_VHDL-1140 FMR_VHDL-1144 FMR_VHDL-1145 FMR_VLOG-079 FMR_VLOG-081 FMR_VLOG-083 FMR_VLOG-087 FMR_VLOG-089 FMR_VLOG-090 FMR_VLOG-091 FMR_VLOG-925 FMR_VLOG-928 FMR_VLOG-929 FMR_ELAB-034 FMR_ELAB-058 FMR_ELAB-059 FMR_ELAB-100 FMR_ELAB-115 FMR_ELAB-116 FMR_ELAB-117 FMR_ELAB-118 FMR_ELAB-125 FMR_ELAB-130 FMR_ELAB-136 FMR_ELAB-145 FMR_ELAB-146 FMR_ELAB-147 FMR_ELAB-149 FMR_ELAB-150 FMR_ELAB-151 FMR_ELAB-153 FMR_ELAB-154 FMR_ELAB-261

DESCRIPTION

The command removes the warning or suppress filter that previously set on one or more simulation-synthesis mismatch messages. The command input should exactly match existing filter to remove.

The command returns status 1 on success and 0 on failure.

EXAMPLES

To remove **-suppress** filter that set on FMR_ELAB-117 for file /vobs/data/rtl/test.sv and line-number 57, use the command as shown below. Note that the remove command should match the set command to get removed.

```
fm_shell (setup)> set_mismatch_message_filter -suppress -file {/vobs/data/rtl/test.sv}
     -line 57 FMR_ELAB-117

fm_shell (setup)> remove_mismatch_message_filter -suppress -file {/vobs/data/rtl/test.sv}
     -line 57 FMR_ELAB-117
```

To remove all the **-suppress** filters set on FMR_ELAB-117, use the command as shown below:

```
fm_shell (setup)> remove_mismatch_message_filter -suppress FMR_ELAB-117
```

To remove all the **-warn** filters that set on mismatch message, use the command as shown below:

```
fm_shell (setup)> remove_mismatch_message_filter -warn -all
```

To remove suppress filter that set on FMR_ELAB-146 that matches file-name test*.v, use the command as shown below:

```
fm_shell (setup)> remove_mismatch_message_filter -suppress -file {*/test*.v} FMR_ELAB-146
```

SEE ALSO

```
set_mismatch_message_filter(2)
report_mismatch_message_filters(2)
```

remove_net

Removes the specified nets.

SYNTAX

```
status remove_net
   [-hier]
   net_list
   -all
```

Data Types

```
net_list list
```

RETURN VALUE

The **remove_net** command returns a status of 1 if it was successful and 0 if it failed.

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-hier

Removes all segments (across hierarchy) of nets in the *net_list*.

net_list

Specifies the names of nets to remove. You can use either object IDs or instance-based paths.

Nets are removed from all instances of a design. If you want to only affect a particular instance-based path, you must uniquify that instance of the design first.

You must specify either the *net_list* or *-all*.

-all

Removes all nets in the current design or instance.

You must specify either the *net_list* or *-all*.

DESCRIPTION

This command removes nets that are specified using the *net_list* argument, or all nets in the current design or instance if the *-all* option is used.

If a net is connected to a pin, the connection is removed.

If the **-hier** option is specified, the segments of the net (across hierarchy) are removed. Any ports now orphaned as a result of this are also removed.

The **-hier** and the **-all** options are mutually exclusive.

You can create nets using the **create_net** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example removes two nets *net1* and *net2* in design r:/WORK/mid.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> remove_net {net1 net2}
```

The following example removes a net *net3* from all instances of the design of r:/WORK/top/m1/b1.

```
fm_shell (setup)> current_instance r:/WORK/top/m1/b1
fm_shell (setup)> remove_net net3
```

SEE ALSO

```
create_net(2)
current_design(2)
current_instance(2)
edit_design(2)
```

remove_net_resolution

Removes resolution function from the specified net.

SYNTAX

```
status remove_net_resolution
   [ -all ]
   [ objectID ]
```

Data Types

objectID	string

Enabled Shell Modes

Setup

ARGUMENTS

-all

Removes resolution function from all the nets. Causes Formaity to select the default resolution function for all the multiply-driven nets.

objectID

Specifies the net from which the resolution function is to be removed. If you specify a name that resolves to more than one net, the resolution function is removed from all of the matching nets.

DESCRIPTION

Use this command to remove the net resolution function of a net with multiple drivers.

One and only one of the two options -all and *objectID* should be specfied. If the command is invoking with both or none of the options, the tool will report an error.

EXAMPLES

The following examples removes the resolution function from the net VDD in the implementation design.

```
fm_shell (setup)> remove_net_resolution \
i:/WORK/dut/VDD
Removed resolution function from net 'i:/WORK/dut/VDD'
1
```

SEE ALSO

```
report_net_resolution(2)
set_net_resolution(2)
```

remove_object

Removes the specified pins, ports, or unlinked cells from a design.

SYNTAX

```
remove_object
   objectID
   [-shared_lib]
   [-type object_type]
```

Data Types

```
objectID string
object_type string
```

Enabled Shell Modes

Setup

ARGUMENTS

objectID

Specifies the pin, port, or unlinked cell to remove.

-shared_lib

Removes the specified objects from the technology libraries.

-type *object_type*

Specifies the object type with the specific object ID. Use this option if the name of the specified design object is associated with more than one object type within a design. Specify one of the following values for the *object_type* argument:

- *cell* for a cell type
- *port* for a port type
- *pin* for a pin type

DESCRIPTION

This command removes the specified pin, port, or unlinked cell from a design.

For example, an implementation design uses black boxes that do not add functionality to the design compared to the reference design. A simple way to make sure the objects do not affect verification is to remove them.

The **remove_object** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example removes a port in the implementation design.

```
fm_shell (setup)> remove_object impl:/WORK/CORE/BUS_KEEPER
1
fm_shell (setup)>
```

SEE ALSO

```
report_designs(2)
```

remove_parameters

Removes specific parameters that are set by using the **set_parameters** command.

SYNTAX

```
status remove_parameters
   [-flatten]
   [-retimed]
   [-resolution]
   [-all_designs]
   [-all_parameters]
   [designID_list]
```

Data Types

```
designID_list string
```

Enabled Shell Modes

Setup (for the **-retimed**, **-resolution**, **-all_parameters** parameters)

ARGUMENTS

-flatten

Removes the flattened parameters.

-retimed

Removes the retimed parameters.

-resolution

Removes the resolution parameters.

-all_designs

Removes parameters from all designs in the Formality environment.

-all_parameters

Removes all three types of parameters.

designID_list

Removes the parameters in the specific design.

DESCRIPTION

This command removes specific parameters that are set by using the **set_parameters** command.

You can remove parameters from all designs, individual designs, or the current design. To remove parameters from all designs, use the *-all_designs* option. To remove parameters from specific designs, specify the design IDs. By default, this command removes parameters from the current design.

Specify the parameters to remove by using any combination of the *-flatten*, *-retimed*, and *-resolution* options. To remove all the three types of parameters, use the *-all_parameters* option.

You must specify at least one parameter type.

When a design is not associated with a flattened parameter, the command verifies hierarchical blocks within their isolated context during verification. If verification fails at this level, the command flattens the design and verifies it again. Remove the flattened parameter to perform block-level verification before flattening the design.

When a design is not associated with the retimed parameter, the command does not account for retiming before verification. For more information about how Formality treats retimed designs, see the *Formality User Guide*.

When a design is not associated with the resolution parameter, the tool determines a signal consensus at nets having multiple drivers where no ports or black boxes are driving the net. For more information about resolution setting, see the *Formality User Guide*.

EXAMPLES

The following example shows how to removes the flattened parameters from the current design in the current container:

```
fm_shell> remove_parameters -flatten
1
fm_shell>
```

SEE ALSO

```
report_parameters(2)
set_parameters(2)
```

remove_port

Removes the specified ports.

SYNTAX

```
status remove_port
   port_list
   -all
```

Data Types

```
port_list list
```

RETURN VALUE

The **remove_port** command returns a status of 1 if it was successful and 0 if it failed.

LICENSE

This command is only available with Formality Ultra using the "Formality-Ultra" license key.

ARGUMENTS

port_list

Specifies the names of ports to remove. You can use either object IDs or instance-based paths.

Ports are removed from all instances of a design. If you want to only affect a particular instance-based path, you must uniquify that instance of the design first.

You must specify either the *port_list* or *-all*.

-all

Removes all ports in the current design or instance. You must specify either the *port_list* or *-all*.

DESCRIPTION

This command removes ports that are specified using the *port_list* argument, or all ports on the current design or instance if the *-all* option is used.

This command also disconnects any nets that are connected to ports that are removed.

You can create ports using the **create_port** command.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example removes two ports *in1* and *in2* on design r:/WORK/mid.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> remove_port {in1 in2}
```

The following example removes a port *in3* from all instances of the design of r:/WORK/top/m1/b1.

```
fm_shell (setup)> current_instance r:/WORK/top/m1/b1
fm_shell (setup)> remove_port in3
```

SEE ALSO

```
create_port(2)
current_design(2)
current_instance(2)
edit_design(2)
```

remove_probe_points

Deselects the two nets in a logic cone set by the **set_probe_points** command.

SYNTAX

```
remove_probe_points
   objectID | -all
```

Data Types

```
objectID string
```

Enabled Shell Modes

Setup
Match
Verify

ARGUMENTS

objectID

Removes the specified object that is a part of a probe pair. The name should specify the absolute path to a unique net in the reference or implementation designs starting with the top design.

-all

Removes the set probe pairs.

DESCRIPTION

This command removes the two nets set by using the **set_probe_points** command in either the reference or the implementation design. The net name should refer to a unique net.

EXAMPLES

```
fm_shell (setup)> **remove_probe_points** *$ref/U1/U2/net2*
Removed probe for 'r:/WORK/top/U1/U2/net2'
1
fm_shell (setup)> **remove_probe_points** *i:/WORK/top/n1*
Removed probe for 'i:/WORK/top/n1'
1
```

SEE ALSO

```
set_probe_points(2)
report_probe_points(2)
report_probe_status(2)
```

remove_resistive_drivers

Removes all resistive drivers including pullups, pulldowns, and bus holders from the Formality environment.

SYNTAX

```
remove_resistive_drivers
```

Enabled Shell Modes

Setup

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command removes all resistive drivers, pullups, pulldowns, bus holders, and similar components from all open containers. These components represent the effects of driver-type attributes specified in the Synopsys libraries or resistive driver constructs in the Verilog libraries.

Use this command to exclude these drivers during subsequent verification.

The **remove_resistive_drivers** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

```
 fm_shell (setup)> remove_resistive_drivers
Removed object 'c:/TGC4000_3.3V_QLM_NOM.DB/IO001/C0'
Removed object 'c:/TGC4000_3.3V_QLM_NOM.DB/IO003/C0'
Removed object 'c:/TGC4000_3.3V_QLM_NOM.DB/PB110/C0'
Removed object 'c:/TGC4000_3.3V_QLM_NOM.DB/PD100/C0'
Removed object 'c:/TGC4000_3.3V_QLM_NOM.DB/PR010/C0'
Removed object 'c:/TGC4000_3.3V_QLM_NOM.DB/PR100/C0'
1
fm_shell (setup)>
```

SEE ALSO

remove_user_match

Removes user-specified matches from the current session.

SYNTAX

```
remove_user_match
   [-type type] | -all
   objectID_list
```

Data Types

```
type string
objectID_list string
```

ARGUMENTS

-type *type*

Specifies the object type for the object. Use this option if the name of the specified design object is associated with more than one object type within the same design. Specify one of the following values for the *type* argument:

- *pin* to specify pin type
- *port* to specify port type
- *net* to specify net type
- *cell* to specify cell type

-all

Removes all user-defined matches.

objectID_list

Specifies a list of design objects that are specified as one of the points in a user-defined match. If you specify a name consisting of a regular expression that resolves to more than one object, the tool reports an error. If the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected. The precedence in this case is pin, port, net, and cell.

DESCRIPTION

This command removes one or more user-defined matches from the current session.

NOTE: Newly created user-defined matches are not applied to designs until the subsequent *match* or *verify* command. Other related points might be matched as a result of a user-defined match. After a user-defined match is applied, you can only undo the match, but not remove it, by using the *undo_match* or *setup* command. This allows you to avoid situations in which the matched and setup states are inconsistent.

The **remove_user_match** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example removes a user-specified match. The *A_T33* port is located in the *impl* container, *WORK* library, and *CORE* design.

```
fm_shell> remove_user_match impl:/WORK/CORE/A_T33
```

The following example is the same as the preceding example except that the *-type* option differentiates the design object from other types having the same name.

```
fm_shell> remove_user_match -type cell impl:/WORK/CORE/A_T33
```

SEE ALSO

```
set_user_match(2)
report_user_matches(2)
undo_match(2)
```

remove_verify_points

Removes a list of user-defined verify points.

SYNTAX

```
remove_verify_points
   [-type ID_type]
   objectID_list
   [-all]
```

ENABLED SHELL MODES

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-type *ID_type*

Identifies the object type for the specified objects. Use this option when the name of the specified design object is associated with more than one object type within the same design. Specify one of the following object types,

- **pin** - To specify the pin type.

- **port** - To specify the port type.
- **net** - To specify the net type.
- **cell**- To specify the cell type.

objectID_list

Specifies design objects to remove from the set of verify points. If you specify a name consisting of a wildcard expression that resolves to more than one object, the operation applies to all the matching objects. If the name resolves to multiple objects with identical names, and you do not specify the object type, only one of these objects is affected. In this case, the precedence is pin, port, net, and cell.

-all

Removes all user-defined verify points.

DESCRIPTION

This command removes compare points from the set of verify points. To remove all user-specified verify points, use the **-all** option.

Verify points affect the subsequent **match** and **verify** commands.

To report the curent verify points, use the following command:

```
report_verify_points
```

EXAMPLES

The following example shows how to remove a point from the set of verify points.

```
fm_shell> remove_verify_points i:/WORK/top/out
```

The following example is similar to the preceding example except that the **-type** option differentiates the design object from other types having the same name.

```
fm_shell> remove_verify_points -type pin i:/WORK/top/out
```

SEE ALSO

report_verify_points(2)

```
set_verify_points(2)
```

rename

Renames or deletes a command.

SYNTAX

```
string **rename**
   old_name
   new_name
```

ARGUMENTS

old_name

Specifies the current name of the command.

new_name

Specifies the new name of the command.

DESCRIPTION

Renames the *old_name* command so that it is now called *new_name*. If *new_name* is an empty string, then *old_name* is deleted. The *old_name* and *new_name* arguments may include namespace qualifiers (names of containing namespaces). If a command is renamed into a different namespace, future invocations of it will execute in the new namespace. The **rename** command returns an empty string as result.

Note that the **rename** command cannot be used on permanent procedures. Depending on the application, it can be used on all basic builtin commands. In some cases, the application will allow all commands to be renamed.

WARNING: **rename** can have serious consequences if not used correctly. When using **rename** on anything other than a user-defined Tcl procedure, you will be warned. The **rename** command is intended as a means to wrap other commands: that is, the command is replaced by a Tcl procedure which calls the original. Parts of the application are written as Tcl procedures, and these procedures can use any command. Commands like **puts**, **echo**, **open**, **close**, **source** and many others are often used within the application. Use **rename** with extreme care and at your own risk. Consider using **alias**, Tcl procedures, or a private namespace before using **rename**.

EXAMPLES

This example renames my_proc to my_proc2:

```
prompt> proc my_proc {} {echo "Hello"}
prompt> rename my_proc my_proc2
prompt> my_proc2
Hello
prompt> my_proc
Error: unknown command 'my_proc' (CMD-005).
```

SEE ALSO

```
define_proc_attributes(2)
```

rename_object

Renames the specified objects in a design.

SYNTAX

```
rename_object
   -file filename
   [-type object_type]
   [-shared_lib]
   [-container container_name]
   [-reverse]
   objectID
   [new_name]
```

Data Types

```
filename string
object_type string
container_name string
objectID string
new_name string
```

Enabled Shell Modes

Setup

ARGUMENTS

-file *filename*

Specifies the file from which to read the rename directives. Each line in the specified file must follow the syntax:

```
[[design_name] object_type] objectID new_name
```

-type *object_type*

Identifies the object type specified by the *objectID* argument. Use this option if the name of the specified design object is associated with more than one object type within the same design. Specify one of the following values for *object_type*:

- *cell* for a cell type
- *port* for a port type

-shared_lib

Renames shared library objects in technology libraries.

-container *container_name*

Specifies the container to apply the rename directives defined by using the *-file* option. The tool examines every library in the container for name changes.

-reverse

Reverses the rename directives in the file.

objectID

Specifies the design object to rename.

new_name

Specifies the new name for the object.

DESCRIPTION

This command renames design objects in the current design. You can specify a single **rename_object** directive or specify a file of directives.

To rename one design object, use the *-type* option. If you are specifying a file of directives, specify the *object_type* argument and optionally a container.

The **rename_object** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example changes port name *ICC* to *ICC_port*.

```
fm_shell (setup)> rename_object -type port ref:/WORK/CORE/CC CC_port
Renamed object 'ref:/WORK/CORE/CC' to 'CC_port'
1
fm_shell (setup)>
```

SEE ALSO

report_aborted_points

Reports the compare points that Formality cannot prove either equivalent or not.

SYNTAX

```
int report_aborted_points
    [-compare_rule]
    [-substring substring]
    [-point_type point_type]
    [-inputs input_type]
    [-status status]
    [-loop]
    [-hard]
    [-list]
    [-last]
    [-never_loads]
    [-always_reset]
```

Data Types

```
substring string
point_type string
input_type string
status string
```

Enabled Shell Modes

Verify

ARGUMENTS

-compare_rule

Reports mapped object names that result from the application of compare rules.

-substring *substring*

Reports points containing the specified substring.

-point_type *point_type*

Reports only the compare points that have characteristics which match the specified type or types. By default, all compare point types are reported. You can specify one or more *point_type* values.

- **bbox_pin** to report all black box pin compare points.
- **bbox_input** to report black box input pin compare points.
- **bbox_inout** to report black box bidirectional pin compare points.
- **loop** to report loop compare points.
- **bbox_net** to report multiply-driven nets that are resolved using black boxes.
- *cut* to report cutpoints.
- **port** to report all port compare points.
- **output** to report output port compare points.
- **inout** to report bidirectional port compare points.
- *directly_undriven_output* to report output and bidirectional ports that do not have a connected net or that have a connected net but do not have a driving pin.
- **reg** to report all register compare points.
- **DFF** to report flip-flop register compare points.
- **LAT** to report latch register compare points.
- **trans** to report transparent latch register compare points.
- **PDCut** to report power domain pin compare points.
- **PGPin** to report power, ground, or power-down function compare points.

-inputs *input_type*

Reports only the compare points with input points that are undriven or unmatched, or both. You can specify the following *input_type* values.

- **undriven** - Report compare points with undriven input points.
- **unmatched** - Report compare points with unmatched input points.

-status *status*

Reports only the compare points that have the polarity specified by the *status* argument. By default, the command reports compare points of any polarity. Specify one or both *status* values.

- **inverted** to report compare points with inverted match polarity.
- **noninverted** to report compare points with noninverted match polarity.

-loop

Reports compare points that are aborted because they are driven by a potentially state-holding asynchronous loop.

-hard

Reports compare points that are aborted because they are hard verification points.

-list

Reports a list of point name pairs, which can be used in further Tcl processing, instead of a formatted table.

-last

Reports a list of point name pairs from the last verification.

-never_loads

Filters the report to only include register compare point pairs that do not synchronously load.

-always_reset

Filters the report to only include register compare point pairs that are constantly reset..

DESCRIPTION

This command reports information about compare points that are aborted during the most recent verification. The tool aborts verification of a compare point when it cannot determine functional equivalence. Points that are located downstream from a potentially state-holding asynchronous loop are not verified and are considered aborted.

This command reports the total number of aborted compare points, followed by the design objects that represent the aborted compare point. For each design object, the report includes the container, the design object type, and the design ID of the design object.

EXAMPLES

The following example reports the compare points that are aborted.

```
prompt> report_aborted_points
1 Aborted compare point:
       0 Loop  (driven by a potentially state-holding asynchronous loop)
       1 Hard  (too complex to solve)

 Hard :  Ref DFF   r:/LIB_DETECTOR/Detector/v_rCMag11_reg[0]
         Impl DFF  i:/WORK/Detector/v_rCMag11_reg[0]

 [BBNet: multiply-driven net
  BBPin: black box pin
  Cut:   cut-point
  DFF:   non-constant DFF register
  DFF0:  constant 0 DFF register
  DFF1:  constant 1 DFF register
  DFFX:  constant X DFF register
```

```
    DFF0X: constrained 0X DFF register
    DFF1X: constrained 1X DFF register
    LAT:   non-constant latch register
    LAT0:  constant 0 latch register
    LAT1:  constant 1 latch register
    LATX:  constant X latch register
    LAT0X: constrained 0X latch register
    LAT1X: constrained 1X latch register
    LATCG: clock-gating latch register
    TLA:   transparent latch register
    TLA0X: transparent constrained 0X latch register
    TLA1X: transparent constrained 1X latch register
    Loop:  cycle break point
    Port:  primary (top-level) port
    Und:   undriven signal cut-point]

  1
```

SEE ALSO

```
report_failing_points(2)
report_matched_points(2)
report_passing_points(2)
report_unmatched_points(2)
report_unverified_points(2)
report_not_compared_points(2)
```

report_always_on_cells

Reports implementation cells that are always on.

SYNTAX

```
report_always_on_cells
   [-domain domain_name]
   [-allowed]
   [-disallowed]
```

Data Types

```
domain_name string
```

ARGUMENTS

-domain *domain_name*

Reports only the always-on cells within the specified UPF power domain.

-allowed

Reports the always-on cells that do not cause verification failure.

-disallowed

Reports always-on cells that cause verification failure. If you specify neither the *-allowed* nor the *-disallowed* options, by default the command reports disallowed always-on cells.

DESCRIPTION

The **report_always_on_cells** command reports the instance path and library/design reference of each allowed or disallowed always-on cell. This command is only available in the verify mode and when the **verification_verify_power_off_states** variable is set to *true*.

Allowed always-on cells are those that are on the following paths:

- Specify paths that drive any primary or backup PG pins
- Logic paths that drive any ISO or SAVE pins
- Paths that connect primary input to primary output wires in the reference design
- The direct drivers of power domain boundaries that are driven by isolation cells in the reference design - cells that are expected to be isolation cells.

EXAMPLES

```
fm_shell (verify)> report_always_on_cells
************************************************
Report         : always_on_cells

Reference      : r:/WORK/ChipTop
Implementation : i:/WORK/ChipTop
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
************************************************
6700 Disallowed always-on cells:
Domain /GPRs/GPRS:
 i:/WORK/ChipTop/GPRs/A_reg_reg_0_ (TCBN65LPHVTWC_CCS/SDFQD1HVT)
 i:/WORK/ChipTop/GPRs/A_reg_reg_10_ (TCBN65LPHVTWC_CCS/SDFQD1HVT)
 i:/WORK/ChipTop/GPRs/A_reg_reg_11_ (TCBN65LPHVTWC_CCS/SDFQD1HVT)
 ...

fm_shell (verify)> report_always_on_cells -allowed
************************************************
Report         : always_on_cells

Reference      : r:/WORK/ChipTop
Implementation : i:/WORK/ChipTop
Version        : G-2012.06
Date           : Wed Jun 6 12:50:30 2012
************************************************
6 Allowed always-on cells:
Domain /Multiplier/GENPP/GENPP:
 i:/WORK/ChipTop/Multiplier/GENPP/U3 (TCBN65LPHVTCGWC/PTINVD4HVT)
 i:/WORK/ChipTop/Multiplier/GENPP/U4 (TCBN65LPHVTCGWC/PTINVD4HVT)
 i:/WORK/ChipTop/Multiplier/GENPP/U5 (TCBN65LPHVTCGWC/PTINVD4HVT)
 i:/WORK/ChipTop/Multiplier/GENPP/U6 (TCBN65LPHVTCGWC/PTINVD4HVT)
 i:/WORK/ChipTop/Multiplier/GENPP/U7 (TCBN65LPHVTCGWC/PTINVD4HVT)
 i:/WORK/ChipTop/Multiplier/GENPP/U8 (TCBN65LPHVTCGWC/PTINVD4HVT)
```

SEE ALSO

```
verification_verify_power_off_states(3)
```

report_analysis_results

Reports the results of the recent analysis.

SYNTAX

```
report_analysis_results
   [-summary]
```

Enabled Shell Modes

Verify

ARGUMENTS

-summary

Reports a summary of the analysis results.

DESCRIPTION

Use this command to report the results of the recent analysis. This is useful after verification when the **verification_run_analyze_points** variable is set to **true**. Also, use this command after restoring a saved session that includes analysis results.

Analysis information is only available in the current verify session. Any action that exits the verify mode removes the analysis results.

SEE ALSO

```
analyze_points(2)
verification_run_analyze_points(3)
```

report_app_var

Shows the application variables.

SYNTAX

```
string report_app_var
   [-verbose]
   [-only_changed_vars]
   [pattern]
```

Data Types

```
pattern      string
```

ARGUMENTS

-verbose

Shows detailed information.

-only_changed_vars

Reports only changed variables.

pattern

Reports on variables matching the pattern. The default is "*".

DESCRIPTION

The **report_app_var** command prints information about application variables matching the supplied pattern. By default, all descriptive information for the variable is printed, except for the help text.

If no variables match the pattern, an error is returned. Otherwise, this command returns the empty string.

If the **-verbose** option is used, then the command also prints the help text for the variable. This text is printed after the variable name and all lines of the help text are prefixed with " #".

The Constraints column can take the following forms:

{val1 ...}

The valid values must belong to the displayed list.

val \<= a

The value must be less than or equal to "a".

val \>= b

The value must be greater than or equal to "b".

b \<= val \<= a

The value must be greater than or equal to "b", and less than or equal to "a".

EXAMPLES

The following are examples of the **report_app_var** command:

```
prompt> report_app_var sh*
Variable                  Value     Type    Default   Constraints
------------------------- --------- ------- --------- ---------------------
sh_continue_on_error      false     bool    false
sh_script_stop_severity   none      string  none      {none W E}
 \.\.\.


prompt> report_app_var sh* -verbose
Variable                  Value     Type    Default    Constraints
------------------------- --------- ------- --------- ---------------------
sh_continue_on_error      false     bool    false
   # Allows source to continue after an error
sh_script_stop_severity   none      string  none      {none W E}
   # Indicates the error message severity level which would cause
   # a script to stop executing before it completes
 \.\.\.
```

SEE ALSO

```
get_app_var(2)
set_app_var(2)
write_app_var(2)
```

report_architecture

Display the architecture and source for each multiplier object ID under the reference design. This command is only available after setup mode.

SYNTAX

```
report_architecture [ -set_architecture |
   -hdlin_multiplier_architecture | -fm_pragma |
   -all | object_ID ]
```

ARGUMENTS

-set_architecture

Display the architecture for multipliers selected via the **set_architecture** command.

-hdlin_multiplier_architecture

Display the architecture for multipliers selected via the **hdlin_multiplier_architecture** TCL variable.

-fm_pragma

Display the architecture for multipliers selected via the Formality source code pragma.

-all

Display the architecture for all multipliers in the reference design. This is the default behavior of the command.

object_ID

Display the architecture for the multiplier at this *object_ID* only.

DESCRIPTION

This command is used to display the architecture selected and the source used to make this selection for each multiplier in the reference design. By default, the architecture and source of all multipliers is displayed, but the command's options can be used to select specific multiplier object_IDs to display based on the source as well as the *object ID*.

This command may only be run after leaving setup mode (i.e. after the **match** command).

EXAMPLES

This example shows displaying the architecture for all multipliers in the design.

```
fm_shell (match)> report_architecture

Arch      Source       Instance Path
csa       set_arch     r:/WORK/test4/mul_28
nbw       hdlin        r:/WORK/test4/mul_30
csa       fm_pragma    r:/WORK/test4/mul_32
wall      fm_pragma    r:/WORK/test4/mul_34
wall      fm_pragma    r:/WORK/test4/mul_40
wall      fm_pragma    r:/WORK/test4/mul_42
nbw       fm_pragma    r:/WORK/test4/mul_44
wall      fm_pragma    r:/WORK/test4/mul_46
```

This example shows displaying the architecture for only multipliers where the hdlin_multiplier_architecture TCL variable was used to select the architecture. Note how the option to the command has been shortened.

```
fm_shell (match)> report_architecture -hdlin

Arch      Source       Instance Path
nbw       hdlin        r:/WORK/test4/mul_30
```

SEE ALSO

set_architecture(2)
enable_multiplier_generation(3)
hdlin_multiplier_architecture(3)
architecture_selection_precedence(3)
dw_foundation_threshold(3)

report_black_boxes

Reports black boxes in the specified designs.

SYNTAX

```
status report_black_boxes
   [-r]
   [-i]
   [-container container_name]
   [-all]
   [-unresolved]
   [-interface_only]
   [-empty]
   [-unread_tech_cell_pins]
   [-set_black_box]
   [designID_list]
```

Data Types

```
container_name string
designID_list string
```

ARGUMENTS

-r

Reports black boxes that are in the default reference container.

-i

Reports black boxes that are in the default implementation container.

-container *container_name*

Reports black boxes that are in the specified container.

-all

Reports all designs that contain black boxes. This is the default behavior of the command.

-unresolved

Reports designs that are referenced by the **read** command, but are not resolved.

-interface_only

Reports designs that contain only the interface. To read in only the interface of a design use the *hdlin_interface_only* variable.

-empty

Reports designs that contain only port declarations.

-unread_tech_cell_pins

Reports designs from technology libraries whose input pins are unread.

-set_black_box

Reports designs that are marked as black boxes by the **set_black_box** command.

designID_list

Specifies the list of designs to report.

DESCRIPTION

This command reports the black boxes that are within the specified designs or containers. By default, the command reports black boxes in the current reference and implementation containers.

To report specific designs, specify the *designID_list* argument. To report designs from specific containers, specify the **-i**, **-r**, or **-container** options.

Each report includes information about:

- Black boxes
- Associated black box types
- Linked or unlinked status for empty designs
- Instance paths of the design inferred as black box

EXAMPLES

The following example shows reports about black boxes in two designs in the reference container. Both designs are in the WORK design library.

```
fm_shell (setup)> report_black_boxes ref:/WORK/lower ref:/WORK/empty
*************************************************
Report         : black_boxes

Reference      : ref:/WORK/top
```

```
Implementation : impl:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
 ____________________________________________________
|                                                    |
|  Legend:                                           |
|          Black Box Attributes                       |
|             s = Set with set_black_box command      |
|             i = Module read with -interface_only    |
|             u = Unresolved design module            |
|             e = Empty design module                 |
|             * = Unlinked design module              |
|            ut = Unread Tech cells pins              |
|____________________________________________________|


##################################################################
####    DESIGN LIBRARY - ref:/WORK
##################################################################
Type Design Name
----  ----------
s     lower

       Instances : 1 of 1
       ------------------------
       ref:/WORK/top/inst1

e     empty

       Instances : 1 of 1
       ------------------------
       ref:/WORK/top/inst2

1
fm_shell>
```

The following example shows reports about all the black box design modules in the implementation and reference containers. Each design is located in the WORK design library.

```
fm_shell (setup)> report_black_boxes
**************************************************
Report         : black_boxes

Reference      : ref:/WORK/top
Implementation : impl:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
Information: Reporting black boxes for current reference and implementation designs. (FM-184)
 ____________________________________________________
|                                                    |
|  Legend:                                           |
|          Black Box Attributes                       |
|             s = Set with set_black_box command      |
|             i = Module read with -interface_only    |
|             u = Unresolved design module            |
|             e = Empty design module                 |
|             * = Unlinked design module              |
|            ut = Unread Tech cells pins              |
|____________________________________________________|

##################################################################
####    DESIGN LIBRARY - impl:/WORK
```

```
#####################################################################
Type Design Name
----  ----------
s     lower

       Instances : 1 of 1
       ------------------------
       impl:/WORK/top/inst1



#####################################################################
####    DESIGN LIBRARY - ref:/WORK
#####################################################################
Type Design Name
----  ----------
s     lower

       Instances : 1 of 1
       ------------------------
       ref:/WORK/top/inst1

e     empty

       Instances : 1 of 1
       ------------------------
       ref:/WORK/top/inst2

1
fm_shell>
```

SEE ALSO

```
remove_black_box(2)
set_black_box(2)
hdlin_interface_only(3)
```

report_cell_list

Reports library cells. This command is available only in the library verification mode.

SYNTAX

```
status report_cell_list
   [-verify]
   [-r | -i]
   [-matched]
   [-unmatched]
   [-passing]
   [-failing]
   [-aborting]
   [-filter wildcard_pattern]
```

Data Types

```
wildcard_pattern string
```

ARGUMENTS

-verify

Reports cells that match in the reference and implementation containers when this option is used before selecting the cells. After you select the cells, this option reports the cells that you selected for verification that matched in both containers. The *-verify* option displays the names of all cells that are selected for verification.

-r

Reports cells that are read into the reference container.

-i

Reports cells that are read into the implementation container.

-matched

Reports cells that are matched in reference and implementation containers.

-unmatched

Reports cells that did not match when you specify this option before you select the cells. After you select the cells, this option reports cells that you tried to select but were not selected for verification because they did not match.

-passing

Reports cells that passed verification.

-failing

Reports cells that failed verification.

-aborting

Reports cells for which the verification was aborted.

-filter *wildcard_pattern*

Specifies a wildcard pattern as a filter. Only the cell names that match the specified pattern are reported.

DESCRIPTION

This command reports library cells that are read by the tool.

- To report cells that are read into the reference container, use the following syntax:

```
report_cell_list -r
```

- To report cells that are selected for verification, use the following syntax:

```
report_cell_list -v
```

Use the **-r** or the **-i** options to report all the cell names that are read after using a **read_*** command.

Use the **-unmatched** and **-verify** options along with the **select_cell_list** command to report cells that are selected for verification.

Use the **-passing**, **-failing**, and **-aborting** options after performing verification to report the cells.

EXAMPLES

This example shows a report about cells in the reference or implementation containers and those that are selected for verification.

```
fm_shell> report_cell_list -r
**************************************************
Report         : cell_list
```

```
Reference      : ref:/WORK/top
Implementation : impl:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

################################################################################
####    Cells in reference container
################################################################################
Myincand
Myand
Mynand
Myor
Mynor

Total cells found: 5

fm_shell> report_cell_list -i
**************************************************
Report         : cell_list

Reference      : ref:/WORK/top
Implementation : impl:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
################################################################################
####    Cells in implementation container
################################################################################
Myincand
Myand
Mynand

Total cells found: 3

fm_shell> report_cell_list -v
**************************************************
Report         : cell_list

Reference      : ref:/WORK/top
Implementation : impl:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
################################################################################
####    Cells selected for verification
################################################################################
Myincand
Myand
Mynand

Total cells found: 3

fm_shell>
```

SEE ALSO

select_cell_list(2)

```
debug_library_cell(2)
```

report_cell_type

Reports the cell type that was set on a technology library cell.

SYNTAX

```
status report_cell_type
   [-all]
   [designID_list]
```

Data Types

```
designID_list string
```

ARGUMENTS

-all

Reports cell-type set by user on all the tech-lib cells.

designID_list

Specifies the list of designs to report.

DESCRIPTION

Reports the cell type associated with the technology library cell or a collection of one or more cells. The types that can be reported are lssd: Level-Sensitive Scan Design retention: Retention register synchronizer: Synchronizer cell multibit: Banked register or Multi-bit register.

EXAMPLES

The following example shows reports about a tech-lib cell with cell-type LSSD.

```
fm_shell (setup)> report_cell_type ref:/WORK/lower
ref:/WORK/lower: SET as LSSD
1
fm_shell>
```

The following example shows reports about a tech-lib cell with cell-type RETENTION.

```
fm_shell (setup)> report_cell_type ref:/WORK/lower
ref:/WORK/lower: SET as RETENTION
1
fm_shell>
```

SEE ALSO

```
remove_cell_type(2)
set_cell_type(2)
```

report_checksum

Reports the checksum value of elaborated top design of a container.

SYNTAX

```
int report_checksum
   [ -container container_name | -r | -i ]
```

Data Types

```
   container_name string
```

ARGUMENTS

-container *container_name*

Reports the checksum value of elaborated top design of specified container.

-r

Reports the checksum value of elaborated top design of reference container.

-i

Reports the checksum value of elaborated top design of implementation container.

DESCRIPTION

Use this command to retrieve the checksum value of elaborated top design of specified linked container. This command generates the checksum by looking at only the elaborated design and library data that is used to define the function to be verified. It can be used on a linked container at any time during the flow to capture the current view of the design and library data. If the checksum value difference is detected between two runs of Formality, it indicates that one or more source data has changed.

If none of the command options are supplied, it generates the checksum of elaborated top design of current container.

EXAMPLES

The following example shows sample checksum report.

```
fm_shell> report_checksum -r
**************************************************
Report         : Checksum
Reference      : r:/WORK/top
Implementation : <None>
Version        : <FM_Version>
Date           : <Date>
**************************************************
The total checksum of the top design r:/WORK/top is -1992367182
1
fm_shell>
```

SEE ALSO

report_clocks

Reports user-specified clocks.

SYNTAX

report_clocks

ARGUMENTS

This command has no arguments

DESCRIPTION

This command reports all user-specified clocks.

EXAMPLES

```
fm_shell (setup)> report_clocks
************************************************
Report         : clocks

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
************************************************
Net Name
--------
HDL:/WORK/CORE/SCLK
1
```

SEE ALSO

```
set_clock(2)
remove_clock(2)
```

report_compare_rules

Reports user-defined compare rules.

SYNTAX

```
report_compare_rules
   [designID]
```

Data Types

```
designID string
```

ARGUMENTS

designID

Specifies the design containing the compare rules.

DESCRIPTION

This command reports compare rules that are defined by using the **set_compare_rule** command.

You can report compare rules added for a specific design by specifying a value for the *designID* argument. If you do not specify a design ID argument, the tool reports user-defined compare rules in the current design by default.

The **report_compare_rules** command reports one line of information for each compare rule. Each line presents the "from" expression followed by the "to" expression. Because many similar compare rules can exist, the tool limits its display to 100 rules. After reporting 100 compare rules, it displays the following message:

```
List of compare rules truncated after 100 compare rules
```

The **report_compare_rules** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example sets a compare rule and then reports the rules. Each line of output represents a single rule:

```
fm_shell> set_compare_rule ref:/WORK/CORE -from "C" -to "B"
Set compare rules for 'ref:/WORK/CORE'
1
fm_shell> report_compare_rules
*************************************************
Report         : compare_rules

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
*************************************************
'C' -> 'B'
1
fm_shell>
```

SEE ALSO

`set_compare_rule(2)`

report_constant_sources

Reports information about the origin of constants on a specified net.

SYNTAX

```
report_constant_sources
   [ netID ... ]
```

ARGUMENTS

netID

Specifies the design net for which you want a report.

DESCRIPTION

This command reports information about the originating source of constants on the specified *netID*s. It traces through the fanin cone of the net until it reaches a constant LOGIC0 or LOGIC1 source, or a net with a user-specified constant on it.

The **report_constant_sources** command produces one line of output for each constant source. The line includes the value, the full object ID, and the object type - net or cell.

The **report_constant_sources** command returns one of the following:

- `0 to indicate failure`
- 1 to indicate success

EXAMPLES

The following example reports on all user-defined constants.

```
fm_shell> report_constant_sources ref:/WORK/UPC/scan_en
Constant   Object
Value      Type     Object Name
--------   ------   -----------
0          cell     ref:/WORK/UPC/LOGIC0
1
fm_shell
```

SEE ALSO

```
report_constants(2)
remove_constant(2)
set_constant(2)
```

report_constants

Reports user-defined constants.

SYNTAX

```
report_constants
   [objectID_list]
```

Data Types

```
objectID_list string
```

ARGUMENTS

objectID_list

Specifies the design objects (net or port) to report.

DESCRIPTION

This command reports constants that are defined by using the **set_constant** command. By default, the tool returns a list of all user-specified constants.

The **report_constants** command generates one line of output for each constant. The line includes the value, the object ID, and the object type, either net or port.

The **report_constants** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example reports all user-defined constants.

```
fm_shell> report_constants
**************************************************
Report         : constants

Reference      : ref:/WORK/UPC
Implementation : impl:/WORK/UPC
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
Constant   Object
Value      Type     Object Name
--------   ------   -----------
0          port     impl:/WORK/UPC/UPC_DATA_11
1          port     ref:/WORK/UPC/UPC_DATA_11
1
fm_shell>
```

SEE ALSO

```
remove_constant(2)
set_constant(2)
```

report_constraint

Reports constraints.

SYNTAX

```
report_constraint
   [-long]
   [constraint_name]
```

Data Types

```
constraint_name string
```

ARGUMENTS

-long

Reports detailed information on all the currently defined constraints.

constraint_name

Specifies the constraint to report.

DESCRIPTION

This command reports the specified constraint. By default, the tool returns information about all constraint types.

When you specify the **-long** option, the tool generates a detailed report on all defined constraints. The tool reports an error when you specify a named constraint.

The report contains the constraint name, type, design, and associated control points on which the constraint is applied. For user-defined constraints, the tool reports the constraint module mappings.

The **report_constraint** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example generates a list of the currently defined constraints.

```
fm_shell> report_constraint
FM_CONSTRAINT_0
FM_CONSTRAINT_1
CPOINT0
CPOINT1
```

The following example returns detailed information about a specific constraint that uses a predefined type.

```
fm_shell> report_constraint FM_CONSTRAINT_0
**************************************************
Report         : constraint

Reference      : ref:/WORK/testcase
Implementation : impl:/WORK/testcase
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

   Constraint: FM_CONSTRAINT_0
   Type:       1hot_init0
   Design:     ref:/WORK/testcase
   Constrained Points...
   (reg)  Q_reg[0]
   (port) PortData[1]
   (reg)  Q_reg[2]
   (reg)  Q_reg[3]
```

The following example returns detailed information about a specific constraint that uses a user-defined type.

```
fm_shell> report_constraint CPOINT0
**************************************************
Report         : constraint

Reference      : ref:/WORK/testcase
Implementation : impl:/WORK/testcase
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

Constraint: CPOINT0
   Type:       odd_parity
   Design:     ref:/WORK/testcase
   Constrained Points...
   (reg)  Q_reg[0]
   (reg)  Q_reg[1]
   (reg)  Q_reg[2]
   (reg)  Q_reg[3]
   Constraint Module Port Map...
```

```
        type:/WORK/my_1hot_4/a <-> ref:/WORK/testcase/Q_reg[0]
        type:/WORK/my_1hot_4/b <-> ref:/WORK/testcase/Q_reg[1]
        type:/WORK/my_1hot_4/c <-> ref:/WORK/testcase/Q_reg[2]
        type:/WORK/my_1hot_4/d <-> ref:/WORK/testcase/Q_reg[3]
        1
```

SEE ALSO

```
  create_constraint_type(2)
  remove_constraint(2)
  remove_constraint_type(2)
  report_constraint_type(2)
  set_constraint(2)
```

report_constraint_type

Reports the constraint types set in a design.

SYNTAX

```
report_constraint_type
   [-long]
   [type_name]
```

Data Types

```
type_name string
```

ARGUMENTS

-long

Generates a detailed report for all the currently defined constraint types.

type_name

Specifies the constraint types to report.

DESCRIPTION

This command reports constraint types, including predefined, external and user-defined, constraint types. See the **set_constraint** command man page for a list of predefined and user-defined types created by using the **create_constraint_type** command.

By default, the tool provides a list of all constraint types in the design. Specify the *type_name* argument to report information about a specific constraint type.

Use the **-long** option to generate a detailed report for all defined constraint types. If you specify a named constraint type with this option, the tool reports an error.

The report includes the names, constraint modules, and constraints that currently reference the constraint types.

The **report_constraint_type** command returns the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example generates a list of defined constraint types.

```
fm_shell> report_constraint_type
**************************************************
Report         : constraint_type

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
  0hot
  0hot_init0
  0hot_init1
  1hot
  1hot_init0
  1hot_init1
  coupled
  mutex
  my_2hot
```

The following example provides detailed information about a predefined constraint type used by a number of constraints.

```
fm_shell> set_constraint 0hot {IN1 IN2 IN3 IN4} $ref
FM_CONSTRAINT_0
fm_shell> set_constraint 0hot {INA INB INC IND} $ref
FM_CONSTRAINT_1
fm_shell> report_constraint_type 0hot
**************************************************
Report         : constraint_type

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
Type:   0hot (predefined)
Constraints...
  FM_CONSTRAINT_0
  FM_CONSTRAINT_1
1
```

The following example provides detailed information about a user-defined constraint type used by a number of constraints.

```
 fm_shell> set_constraint parity {IN1 IN2 IN3 IN4} \
           -map {A=IN1 B=IN2 C=IN3 D=IN4} -name CPOINT0 $ref
fm_shell> report_constraint_type parity
**************************************************
Report         : constraint_type

Reference      : ref:/WORK/parity
Implementation : impl:/WORK/parity
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

Type:   parity (custom)
Design: type:/WORK/parity
Constraints...
  CPOINT0
1
```

SEE ALSO

```
create_constraint_type(2)
remove_constraint(2)
remove_constraint_type(2)
report_constraint(2)
set_constraint(2)
```

report_containers

Reports containers in the current Formality environment.

SYNTAX

```
status report_containers
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports a list of containers in the current Formality environment.

EXAMPLES

The following example creates a report about all containers in the Formality environment.

```
fm_shell> report_containers
**************************************************
Report         : compare_rules

Reference      : ref:/
Implementation : impl:/
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
  Containers:
  impl
  ref
1
fm_shell>
```

SEE ALSO

```
create_container(2)
current_container(2)
read_container(2)
```

report_cutpoints

Reports the cutpoints that are set by using the **set_cutpoint** command.

SYNTAX

```
status report_cutpoints
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports the user-specified cutpoints in the current design.

EXAMPLES

```
fm_shell (setup)> report_cutpoints
*************************************************
Report         : cutpoints

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
*************************************************
Object
 Type     Object Name
------    -----------
pin       HDL:/WORK/CORE/MUX_OUT_BLK/DATA[10]
```

```
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA[11]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA[12]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA[2]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA_OUT[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA_OUT[2]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/DATA_OUT[3]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/ENABLE
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/MUX_OUT[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/MUX_OUT[2]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/REG_DATA[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/REG_DATA[2]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/REG_DATA[3]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/SEL[0]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/SEL[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[2]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[3]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[4]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[1]
      pin        HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[2]
      net        OPT:/WORK/CORE/PC[1]
      1
```

SEE ALSO

```
set_cutpoint(2)
remove_cutpoint(2)
create_cutpoint_blackbox(2)
```

report_design_libraries

Reports design libraries in the current Formality environment.

SYNTAX

```
status report_design_libraries
   [item_list]
```

Data Types

```
   item_list string
```

ARGUMENTS

item_list

Specifies the containers and design libraries to report. The list can have:

- *container_name*: Specifies the container name. Do not use the colon character (:). The **report_design_libraries** command does not support the current container.
- *libraryID*: Specifies a design library name. You can use the (*) wildcard character.

DESCRIPTION

This command reports design libraries in the Formality environment. The report does not include information about technology libraries.

You can report information about specific design libraries and containers by specifying *item_list* arguments. By default, the tool reports information about the design libraries in the current Formality environment.

Each line of the report contains the number of designs in the design library and their names.

EXAMPLES

The following example reports information about all the design libraries loaded in the Formality environment.

```
fm_shell> report_design_libraries
**************************************************
Report         : design_libraries

Reference      : ref:/WORK
Implementation : impl:/my_lib
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

   Number of
   Designs     Design Library
   ---------   --------------
      10       impl:/my_lib
      10       ref:/WORK
1
fm_shell>
```

The following example reports on the design library named WORK loaded in the reference container.

```
fm_shell> report_design_libraries ref:/WORK
**************************************************
Report         : design_libraries

Reference      : ref:/WORK
Implementation : impl:/my_lib
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
   Number of
   Designs     Design Library
   ---------   --------------
      10       ref:/WORK
1
fm_shell>
```

SEE ALSO

```
report_containers(2)
report_designs(2)
report_libraries(2)
```

report_designs

Reports information on designs.

SYNTAX

```
status report_designs
   [item_list]
```

Data Types

```
item_list string
```

ARGUMENTS

item_list

Specifies the containers, libraries, and designs to report. The list can have:

- *container_name* - specifies a container name. Do not use the colon character (:).
- *libraryID* - specifies a design library. You can use the wildcard character (*).
- *designID* - specifies a design. You can use the wildcard character (*).

DESCRIPTION

Use this command to report information about the designs in the Formality environment. You can specify container names, library IDs, and design IDs.

The report includes the following information:

- Number of cells (linked and unlinked)
- Number of ports
- Number of nets

- Resolution type
- Setting for the flatten parameter
- Setting for the retimed parameter

EXAMPLES

The following example shows how to report information about the designs in the container *spec*.

```
fm_shell> report_designs spec
**************************************************
Report         : designs

Reference      : ref:/WORK
Implementation : spec:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

 ____________________________________________________________________________
|                                                                            |
|  Legend:                                                                   |
|          Statistics                  Attributes                            |
|            c = number of cells         c = CONSENSUS resolution type       |
|                linked(unlinked)        x = BLACKBOX resolution type        |
|            p = number of ports         a = AND resolution type             |
|            n = number of nets          o = OR resolution type              |
|                                        f = flatten                         |
|                                        r = retimed                         |
|____________________________________________________________________________|
############################################################################
####    DESIGN LIBRARY - spec:/WORK
############################################################################
Design Name         Statistics          Attributes     Verification status
-----------         ----------          ----------     -------------------
impl                c0(0) p0 n0         c
myand2              c1(0) p3 n3         c
myand2_reg          c7(0) p4 n10        c
myopt1              c6(0) p3 n8         c
myopt2              c0(0) p0 n0         c
myor2               c0(0) p0 n0         c
myor2_reg           c0(0) p0 n0         c
myreg               c7(0) p3 n9         c
spec                c8(0) p4 n11        c
1
fm_shell>
```

This example shows how to report information about the design library WORK in the container *impl*. The example also reports on the designs in the container *spec*.

```
fm_shell> report_designs {impl:/WORK/myopt2 spec}
**************************************************
Report         : design_libraries

Reference      : ref:/WORK
Implementation : spec:/my_lib
```

```
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
__________________________________________________________________________
|                                                                          |
|  Legend:                                                                 |
|          Statistics                  Attributes                          |
|            c = number of cells         c = CONSENSUS resolution type    |
|                linked(unlinked)        x = BLACKBOX resolution type     |
|            p = number of ports         a = AND resolution type          |
|            n = number of nets          o = OR resolution type           |
|                                        f = flatten                      |
|                                        r = retimed                      |
|__________________________________________________________________________|
############################################################################
####    DESIGN LIBRARY - impl:/WORK
############################################################################
Design Name            Statistics         Attributes     Verification status
-----------            ----------         ----------     -------------------
myopt2                 c7(0) p4 n9        c              Passing
############################################################################
####    DESIGN LIBRARY - spec:/WORK
############################################################################
Design Name       Statistics             Attributes     Verification status
-----------       ----------             ----------     -------------------
impl              c0(0) p0 n0            c
myand2            c1(0) p3 n3            c
myand2_reg        c7(0) p4 n10           c
myopt1            c6(0) p3 n8            c
myopt2            c0(0) p0 n0            c
myor2             c0(0) p0 n0            c
myor2_reg         c0(0) p0 n0            c
myreg             c7(0) p3 n9            c
spec              c8(0) p4 n11           c
1
fm_shell>
```

SEE ALSO

```
report_containers(2)
report_design_libraries(2)
report_libraries(2)
find_designs(2)
```

report_diagnosed_matching_regions

Reports matching regions for the previous diagnosis.

SYNTAX

```
int report_diagnosed_matching_regions
   [-expand mode]
```

Data Types

```
mode string
```

Enabled Shell Modes

Verify

ARGUMENTS

-expand *mode*

Reports information about the technology library and DesignWare components.

The *mode* argument specifies the technology library cell expansion mode.

- **auto** - Includes details of the cells containing multiple sequential elements, feedback loops, or other logic that can modify the output of the cell. This is the default mode.
- **true** - Includes details of only the technology library and DesignWare component cells. Only the elements that are part of the diagnosed region are listed.
- **false** - Excludes details of the technology library and DesignWare component cells.

DESCRIPTION

This command reports matching regions for the previous diagnosis. Matching regions are drivers in the matched design and are identified in the undiagnosed design. For example, if you diagnosed the implementation design, matching regions are identified in the reference design.

The report includes recommended and alternative matching regions, and the type of driver.

EXAMPLES

The following example shows a list of matching regions for a diagnosis with three matching regions:

```
fm_shell> report_diagnosed_matching_regions
**************************************************
Report         : diagnosed_matching_regions

Reference      : ref:/WORK
Implementation : impl:/my_lib
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

3 distinct matching regions detected in reference.

Recommended matching region:
    Prim    fm_m1876/C1
    Prim    C782
    Port    e_cd_en

Alternate matching regions:
    Prim    fm_m171f/C1
    Prim    fm_m1876/C1
    Port    e_cd_en
```

SEE ALSO

diagnose(2)

report_dont_cuts

Reports the dont-cut points that are set using the **set_dont_cut** command.

SYNTAX

```
status report_dont_cut
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports the user-specified dont-cut points in the current design.

EXAMPLES

```
fm_shell (setup)> report_dont_cut
*************************************************
Report         : dont_cut

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06-Beta4
Date           : Mon Apr 23 16:43:04 2012
*************************************************

Object
 Type     Object Name
------    -----------
```

```
    pin       r:/WORK/top/block/IN

    1
```

SEE ALSO

```
set_dont_cut(2)
remove_dont_cut(2)
```

report_dont_match_points

Reports the dont_match points that are set using the **set_dont_match_point** command.

SYNTAX

```
status report_dont_match_points
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **report_dont_match_points** command reports the objects that are defined as dont_match using the **set_dont_match_point** command.

EXAMPLES

The following example shows a report of three dont_match points.

```
fm_shell (setup)> report_dont_match_points
**************************************************
Report         : dont_match_points

Reference      : ref:/mp_gt
Implementation : impl:/mp_gt
Version        : H-2013.03
Date           : Mon Mar 11 12:46:37 2013
**************************************************
```

```
      Don't match points:

          (Port)  impl:/mp_gt/IBM_CMOS_GApwr.db/CORE/mydesign
          (Port)  ref:/mp_gt/IBM_CMOS_GApwr.db/CORE/mydesign
          (Reg)   impl:/mp_gt/IBM_CMOS_GApwr.db/CORE/STATE_2
```

SEE ALSO

```
remove_dont_match_points(2)
set_dont_match_points(2)
report_matched_points(2)
report_unmatched_points(2)
```

report_dont_verify_points

Reports dont_verify points that are set by using the **set_dont_verify_point** command.

SYNTAX

```
status report_dont_verify_points
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports the compare points that are defined as dont_verify by using the **set_dont_verify_point** command.

The dont_verify points are not applied to the designs until you run the **match** or the **verify** commands. After using the commands, use the **report_matched_points -point_type dont_verify** and **report_unmatched_points -point_type dont_verify** commands to see which compare points are marked as dont_verify.

EXAMPLES

The following example shows a report of three dont_verify points.

```
fm_shell (setup)> report_dont_verify_points
***********************************************
Report         : dont_verify_points
```

```
Reference      : ref:/mp_gt
Implementation : impl:/mp_gt
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

Dont verify points:

    (Port)  impl:/mp_gt/IBM_CMOS_GApwr.db/CORE/mydesign
    (Port)  ref:/mp_gt/IBM_CMOS_GApwr.db/CORE/mydesign
    (Reg)   impl:/mp_gt/IBM_CMOS_GApwr.db/CORE/STATE_2

fm_shell (setup)>
```

SEE ALSO

```
remove_dont_verify_points(2)
report_matched_points(2)
report_unmatched_points(2)
set_dont_verify_points(2)
```

report_dp_int_round

Reports any rounding information for multipliers and how it was applied.

SYNTAX

`report_dp_int_round`

ENABLED SHELL MODES

Setup

DESCRIPTION

Reports any rounding information associated with multiplers and how it was applied.

EXAMPLES

```
fm_shell (guide)> report_dp_int_round
report_dp_int_round

Instances with rounding information
   (* if rounding was not used)
-----------------------------------
    16  14  r:/WORK/top/U1/mult_x_2

1
```

SEE ALSO

```
set_dp_int_round(2)
remove_dp_int_round(2)
```

report_eco_impact

Reports the impact of the ECO on the original netlist.

SYNTAX

```
integer report_eco_impact
   [-scan str]
   [-unread]
   [-size]
   [-all]
```

Data Types

```
str string
```

Enabled Shell Modes

Match Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

ARGUMENTS

-scan *str*

Reports the impact the ECO has had on the scan chain. New registers introduced by the ECO are excluded from the scan chain. Replacements for banked register bits are also excluded from the scan chain, however the original banked bit remains on the scan chain but is unread on the functional path. ECO's performed on scan-shift registers result in a broken scan chain.

Specify one of the following values for the *str* argument:

- *new* - Report newly introduced registers.

- *multibit* - Report patched multibit registers.
- *shift* - Report shift registers.
- *all* - Report all of the above.

-unread

Reports registers that are left with no readers after the ECO.

-size

Reports the ECO patch size.

-all

Produces all of the above reports (-scan all, -unread and -size)

DESCRIPTION

The **report_eco_impact** command reports on the impact that the ECO patch has on the original netlist.

It should be run in **match** or **verify** modes after successfully generating an ECO patch.

An ECO patch can cause the addition (removal) of new (existing) registers, which can impact the scan chain of the design because it hasn't been rewired to accommodate the additional (removed) register. The **-scan** option of this command can be used to summarize the scan chain impacts of the ECO patch.

Registers that are left without any readers after the ECO can be obtained using the **-unread** option.

Use the **-size** option for a summary of the total number of cells/nets/ports added and removed.

EXAMPLES

The following command produces all of the ECO impact reports:

```
verify> report_eco_impact -all
**************************************************
Report         : report_eco_impact
                  -all

Version        : M-2016.12-ALPHA-160929
Date           : Thu Sep 29 14:34:50 2016

**************************************************

The following registers were introduced by the ECO and are not on the scan chain:

   New  DFF  onet:/WORK/test/iECO_region1_R4_reg
```

```
The following registers were patched by the ECO and are no longer shift-registers.
As a result the scan chain is broken:

 Shift  DFF  onet:/WORK/test/R2_reg

ECO patch (fm_eco_region.patch.tcl) introduces 6 cells, 6 nets, 0 ports and
removes 3 cells, 2 nets.


1
```

SEE ALSO

report_edits

Reports the recorded edit commands.

SYNTAX

```
report_edits
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command reports the recorded edit commands (such as **create_net** and **disconnect_net**). The edit commands shown are the ones that can be used by Design Compiler or IC Compiler to replicate the edits performed in Formality.

Use the command **record_edits** to enable and disable recording.

All edit commands are shown except ones that occurred while recording was turned off and ones that are reverted using the **undo_edits** command.

SEE ALSO

```
create_net(2)
disconnect_net(2)
record_edits(2)
remove_net(2)
write_edits(2)
```

report_electrical_checks

Checks the connectivity of the circuit.

SYNTAX

```
report_electrical_checks
   [ -unconnected_ports ]
   [ -unconnected_pins ]
   [ -output_pins_tied_to_constants ]
   [ -multiply_driven_pins ]
   [ -multiply_driven_ports ]
   [ -unread_nets ]
   [ -undriven_nets ]
   [ -unreachable_cells ]
   [ -uninstantiated_designs ]
   [ -list ]
   [ -edits ]
   [ design_list ]
```

Data Types

```
design_list list of strings
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-unconnected_pins

Checks for pins that are not connected. All pins are checks on primitive gates and tech cells. Only IN and IN/OUT pins are checks on other cells.

-unconnected_ports

Checks for ports that are not connected.

-output_pins_tied_to_constants

Checks for OUT and IN/OUT pins that are tied directly to a constant net.

-multiply_driven_pins

Checks for pins driven by multiple sources.

-multiply_driven_ports

Checks for ports driven by multiple sources.

-unread_nets

Checks for nets without a receiver.

-undriven_nets

Checks for nets without a driver.

-unreachable_cells

Checks for cells that cannot be reached from cell inputs or constant nets.

-uninstantiated_designs

Checks for designs that are not instantiated by at least one cell.

-list

Report the results of the checks as a Tcl list. You can use this list in further Tcl processing.

-edits

Perform the checks in the designs affected by edit commands like **create_net** and **connect_net**.

design_list

Performs the checks in these designs.

This option can be combined with the \fi-edits option. If neither options are use, the command checks all implementation designs.

DESCRIPTION

This command performs a number of checks of the connectivity of the circuit. The format is either a report of the objects that failed the checks, or, if the **-list** option is used, a Tcl list.

The Tcl output is a list of alternating keys and values. The values are lists of objects.

```
UNCONNECTED_PINS <PINS>
UNCONNECTED_PORTS <PORTS>
MULTIPLY_DRIVEN_PINS <PORTS>
```

```
MULTIPLY_DRIVEN_PORTS <PORTS>
OUTPUT_PINS_TIED_TO_CONSTANT <PINS>
UNREAD_NETS <NETS>
UNDRIVEN_NETS <NETS>
UNREACHABLE_CELLS <CELLS>
UNINSTANTIATED_DESIGNS <DESIGNS>
```

By default, the **report_electrical_checks** command performs all the checks on all implementation designs. However, you can limit the checks it performs via the command options, and you can specify the designs you want to check.

EXAMPLES

The following example performs all checks on all implemenetation designs:

```
fm_shell> report_electrical_checks
Processing design: i:/WORK/M_RTL_ADD_UNS_4

Processing design: i:/WORK/alt_bot1
 Design not instantiated: i:/WORK/alt_bot1

Processing design: i:/WORK/alt_bot2

Processing design: i:/WORK/bot
 Output pins tied to constants in design: i:/WORK/bot
  out  i:/WORK/bot/C0/OUT
 Unread nets in design: i:/WORK/bot
  i:/WORK/bot/fm_N7[10]
  i:/WORK/bot/fm_N7[11]
  i:/WORK/bot/fm_N7[8]
  i:/WORK/bot/fm_N7[9]
 Undriven nets in design: i:/WORK/bot
  i:/WORK/bot/fm_N7[4]
  i:/WORK/bot/fm_N7[5]
  i:/WORK/bot/fm_N7[6]

Processing design: i:/WORK/mid
 Unconnected ports on design: i:/WORK/mid
  in  i:/WORK/mid/mck
 Unconnected pins on cells in design: i:/WORK/mid
  in  i:/WORK/mid/MY_PRIM/IN1
  in  i:/WORK/mid/MY_PRIM/IN2
  out  i:/WORK/mid/MY_PRIM/OUT
 Undriven nets in design: i:/WORK/mid
  i:/WORK/mid/mck
 Unreachable cells in design: i:/WORK/mid
  i:/WORK/mid/MY_PRIM

Processing design: i:/WORK/top
 Unread nets in design: i:/WORK/top
  i:/WORK/top/i5
 Undriven nets in design: i:/WORK/top
  i:/WORK/top/o5
```

SEE ALSO

```
create_net(2)
connect_net(2)
```

report_error_candidates

Reports error candidates for the previous diagnosis.

SYNTAX

```
status report_error_candidates
   [-compare_point compare_point]
   [-design design]
   [-percentage percentage]
   [-techlib]
   [-expand type]
```

Data Types

```
compare_point string
design string
percentage integer
type string
```

Enabled Shell Modes

Verify

ARGUMENTS

-compare_point *compare_point*

This option is obsolete.

-design *design*

This option is obsolete.

-percentage *percentage*

Reports error candidates that are above a specific percentage. Specify a value between 0 to 100.

-techlib

This option is obsolete.

-expand *type*

Reports detailed information on the technology library and DesignWare components. Specify one of the following values for the *type* argument:

- **false** - Excludes details of technology library and DesignWare component cells from the report.
- **auto** - Includes details of only the cells containing multiple sequential elements, feedback loops, or other logic that can modify the output value of the cell in the report. This is the default mode.
- **true** - Includes details of only the technology library and DesignWare component cells in the report. Only the elements that are part of the error candidate are listed.

DESCRIPTION

This command reports error candidates for the previous diagnosis. Error candidates are driver objects in the design whose state during verification might fail the verification.

If the design has multiple errors, diagnosis might identify error candidates that are comprised of multiple error drivers. The **report_error_candidates** command reports the number of distinct errors and error candidates. Diagnosis might find close alternative error candidates that can also be used to correct the failures, in addition to the recommended candidate.

The report contains a recommended candidate, an optional list of alternative candidates, and the type of the driver.

EXAMPLES

The following example shows a list of error candidates for a single error diagnosis. The command reports two error candidates, each of type cell:

```
fm_shell (verify)> report_error_candidates
**************************************************
Report         : error_candidates

Reference      : ref:/WORK
Implementation : impl:/my_lib
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

Single error detected in implementation.

Recommended error candidate:
    Cell    U4072

Alternate error candidates:
 1. Cell    U4071

1
```

The following example shows a list of four error candidates, with two drivers each.

```
    fm_shell (verify)> **report_error_candidates**
    **************************************************
    Report         : error_candidates

    Reference      : ref:/WORK
    Implementation : impl:/my_lib
    Version        : G-2012.06
    Date           : Wed Jun 6 12:46:37 2012
    **************************************************
    2 distinct errors detected in implementation.

    Recommended error candidate:
        U5036
        U4652

    Alternate error candidates:
     1. bd_cpu_end_reg
        U4652

     2. U5036
        bd_underflow_prev_reg

     3. bd_cpu_end_reg
        bd_underflow_prev_reg

    1
```

SEE ALSO

```
diagnose(2)
report_diagnosed_matching_regions(2)
```

report_factor_points

Reports factoring variables that are set by using the **set_factor_point** command.

SYNTAX

```
status report_factor_points
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports factoring variables that are set by using the **set_factor_point** command.

EXAMPLES

```
fm_shell (setup)> report_factor_points
**************************************************
Report         : factor_points

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
Object
 Type     Object Name
------    -----------
port      r:/WORK/top/data1
port      r:/WORK/top/data2
port      r:/WORK/top/data3

1
```

SEE ALSO

```
set_factor_point(2)
remove_factor_point(2)
```

report_failing_points

Reports compare points that fail verification.

SYNTAX

```
int report_failing_points
   [-compare_rule]
   [-substring substring]
   [-point_type point_type]
   [-matched]
   [-unmatched]
   [-status status]
   [-inputs input_type]
   [-list]
   [-last]
   [-never_loads]
   [-always_reset]
```

Data Types

```
substring string
point_type string
status string
input_type string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-compare_rule

Reports object names that are mapped because of the application of compare rules.

-substring substring

Reports the compare points that contain the specified substring.

-point_type *point_type*

Reports the compare points with characteristics that match the specified types. By default, all compare point types are included. Specify the following point types:

- *bbox_pin* - Reports black box pin compare points
- *bbox_input* - Reports black box input pin compare points
- *bbox_inout* - Reports black box bidirectional pin compare points
- *loop* - Reports loop compare points
- *bbox_net* - Reports multiply-driven nets that are resolved using black boxes
- *cut* - Reports cutpoints
- *port* - Reports port compare points
- *output* - Reports output port compare points
- *inout* - Reports bidirectional port compare points
- *directly_undriven_output* - Reports output and bidirectional ports that do not have a connected net, or have a connected net but do not have a driving pin
- *reg* - Reports register compare points
- *DFF* - Reports flip-flop register compare points
- *LAT* - Reports latch register compare points
- *trans* - Reports transparent latch register compare points
- *PDCut* - Reports power domain pin compare points
- *PGPin* - Reports power, ground, or power-down function compare points

-matched

Reports the matched design objects.

-unmatched

Reports the unmatched design objects.

-status *status*

Reports only the compare points that have the specified polarity. By default, the command reports compare points of any polarity. Specify one or both of the following *status* values:

- *inverted* - to report compare point pairs with inverted polarity.
- *noninverted* - to report compare point pair with noninverted polarity.

-inputs *input_type*

Reports only the compare points with undriven or unmatched input points. Specify one or both of the following input types.

- *undriven* - to report compare points with undriven input points.
- *unmatched* - to report compare points with unmatched input points.

-list

Reports a list of compare point pairs.

-last

Reports a list of compare point pairs from the previous verification.

-never_loads

Filters the report to only include register compare point pairs that do not synchronously load.

-always_reset

Filters the report to only include register compare point pairs that are constantly reset.

DESCRIPTION

This command reports compare points that failed the previous verification. A failing compare point consists of two design objects that are not equivalent or a compare point having an unmatched design object.

The command lists the number of failing compare points followed by paired design objects for each failing compare point. Each item in the report lists the design object, the design that contains the failing compare point (implementation or reference design), the design object type, and the design ID. When you use the **-unmatched** option, the number of failing compare points is followed by the unpaired design objects.

EXAMPLES

The following example shows a failing point report.

```
fm_shell> report_failing_points
**************************************************
Report         : failing_points

Reference      : OPT:/WORK/CORE
Implementation : SCAN:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

20 Failing compare points (20 matched, 0 unmatched):

  Ref DFF        OPT:/WORK/CORE/REG_BLK/INT_REG_reg[10]
```

```
 Impl DFF        SCAN:/WORK/CORE/REG_BLK/INT_REG_reg[10]

 Ref DFF        OPT:/WORK/CORE/REG_BLK/INT_REG_reg[11]
 Impl DFF    (-) SCAN:/WORK/CORE/REG_BLK/INT_REG_reg[11]

 Ref DFF        OPT:/WORK/CORE/REG_BLK/INT_REG_reg[12]
 Impl DFF    (-) SCAN:/WORK/CORE/REG_BLK/INT_REG_reg[12]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][10]
 Impl DFF    (-) SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][10]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][11]
 Impl DFF    (-) SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][11]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][12]
 Impl DFF    (-) SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][12]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][1]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][1]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][3]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][3]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][4]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][4]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][5]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][5]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][6]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][6]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][7]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][7]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][8]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][8]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[2][9]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[2][9]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[4][10]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[4][10]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[4][11]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[4][11]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[4][12]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[4][12]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[4][1]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[4][1]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[4][3]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[4][3]

 Ref DFF        OPT:/WORK/CORE/STACK_BLK/MEM_reg[4][4]
 Impl DFF        SCAN:/WORK/CORE/STACK_BLK/MEM_reg[4][4]

[BBNet: multiply-driven net
 BBPin: black box pin
 Cut:   cut-point
 DFF:   non-constant DFF register
 DFF0:  constant 0 DFF register
```

```
       DFF1:  constant 1 DFF register
       DFFX:  constant X DFF register
       DFF0X: constrained 0X DFF register
       DFF1X: constrained 1X DFF register
       LAT:   non-constant latch register
       LAT0:  constant 0 latch register
       LAT1:  constant 1 latch register
       LATX:  constant X latch register
       LAT0X: constrained 0X latch register
       LAT1X: constrained 1X latch register
       LATCG: clock-gating latch register
       TLA:   transparent latch register
       TLA0X: transparent constrained 0X latch register
       TLA1X: transparent constrained 1X latch register
       Loop:  cycle break point
       Port:  primary (top-level) port
       Und:   undriven signal cut-point]

     1
```

SEE ALSO

```
report_aborted_points(2)
report_matched_points(2)
report_passing_points(2)
report_unmatched_points(2)
report_unverified_points(2)
report_not_compared_points(2)
```

report_fsm

Reports information on finite state machines (FSM) in a design.

SYNTAX

```
status report_fsm
   [-name FSM_name | designID]
```

Data Types

```
   FSM_name string
   designID string
```

ARGUMENTS

-name *FSM_name*

Reports state information of the specific FSM.

designID

Specifies the design for which FSM information is reported. By default, the command reports FSM information in the current design.

DESCRIPTION

This command reports FSM state information in the current design by default, or in a specific design. The report includes the FSM name, state vector flip-flop names and ordering, and the encodings.

EXAMPLES

The following example reports the state encodings for the design impl:/WORK/CORE/FSM.

```
fm_shell> report_fsm impl:/WORK/CORE/FSM
************************************************
Report         : fsm

Reference      : ref:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
************************************************
FM_FSM_0
State Vector:
CURRENT_STATE_reg[1]
CURRENT_STATE_reg[0]
State Encodings:
S1
10
S2
01
S3
11
SO
00
1
fm_shell>
```

The following example is the same as the preceding example except that it uses the *-name* option to specify the FSM name.

```
fm_shell> report_fsm -name FM_FSM_0
************************************************
Report         : fsm

Reference      : ref:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
************************************************
FM_FSM_0
State Vector:
CURRENT_STATE_reg[1]
CURRENT_STATE_reg[0]
State Encodings:
S1
10
S2
01
S3
11
SO
00
1
fm_shell>
```

SEE ALSO

```
read_fsm_states(2)
```

```
set_fsm_encoding(2)
set_fsm_state_vector(2)
```

report_guidance

Reports the name and the summary of contents of the current automated Setup Verifiction for Formality (SVF) file(s).

SYNTAX

```
report_guidance
   [-to filename]
   [-pack]
   [-datapath]
   [-long]
   [-short]
   [-source]
   [-summary]
```

Data Types

```
filename string
```

ARGUMENTS

-to *filename*

Specifies the name of the output file. If an intermediate automated setup file for verification netlist is referenced in the automated setup file, the command stores the unencrypted versions of the referenced netlists in a directory prefixed with fmsvf.

-pack

Creates a compressed file containing the output of the **report_guidance** command. Use this option with the **-to** option.

-datapath

Reports a summary of the status of datapath transformations.

-long

Reports detailed information on the status of datapath transformations. Use this option with the **-datapath** option.

-short

Skip individual guide summary in report

-source

Returns the source of the most recently loaded SVF file

-summary

Reports a summary of the automated setup file. This option is the default behavior.

DESCRIPTION

This command reports the name and the summary of the contents of the current Setup Verification for Formality (SVF) files. The command overwrites any existing file with the file specified by the **-to** option.

EXAMPLES

```
fm_shell (setup)> report_guidance
**************************************************
Report         : guidance

Reference      : ref:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
SVF is not set.
1

fm_shell (setup)> set_svf myfile.svf
SVF set to 'myfile.svf'.
1

fm_shell (setup)> report_guidance -to mynewfile.svf
**************************************************
Report         : guidance

Reference      : ref:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
SVF files read:
  myfile.svf

SVF files produced:
  mynewfile.svf/
    svf.txt
    d1/
```

Note that if there are references to intermediate encrypted netlists in the automated setup file for verification such as
guide_architecture_netlist -file { dwsvf_7839-0/dwarchs_0.v.e } { lsi_10k }
then the decrypted automated setup file for verification has new references to the decrypted netlists such as **guide_architecture_netlist -file { fmsvf_7839-0/dwarchs_0.v } { lsi_10k }**

```
fm_shell (setup)> report_guidance -summary
**************************************************
Report         : guidance

Reference      : ref:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

                                         Status
Command                 Accepted   Rejected Unsupported Unprocessed Total
----------------------------------------------------------------------------
architecture_netlist:          1          0           0           0       1
change_names        :          1         37           0           0      38
datapath            :          0          2           0           0       2
inv_push            :         19          0           0           0      19
reg_constant        :          1          0           0           0       1
transformation
   map              :         66          0           0           0      66
   merge            :          2          0           0           0       2
   share            :          8          0           0           0       8

Note: If verification succeeds you can safely ignore unaccepted guidance commands.

SVF files read:
  myfile.svf

SVF files produced:
  formality_svf/
    svf.txt
    d1/
```

SEE ALSO

```
set_svf(2)
remove_guidance(2)
find_svf_operation(2)
report_svf_operation(2)
```

report_hdlin_mismatches

Reports the RTL simulation or synthesis mismatches that occured during design linking.

SYNTAX

```
report_hdlin_mismatches
   [-summary | -verbose]
   [-reference]
   [-implementation]
   [-container containerName]
```

Data Types

```
containerName string
```

Enabled Shell Modes

Verify

ARGUMENTS

-summary

Reports a summary of mismatches.

-verbose

Reports detailed information about mismatches, including a list of all locations. This is the default behavior.

-reference

Reports mismatches in the current reference container.

-implementation

Reports mismatches in the current implementation container.

-container *containerName*

Reports mismatches in the specified container.

DESCRIPTION

This command reports information about simulation or synthesis mismatches that occur during the setup of the specified containers.

EXAMPLES

```
fm_shell (verify)> report_hdlin_mismatches
[Verbose report of the current container]

fm_shell (verify)> report_hdlin_mismatches -r -i -s
[Summary report of both the reference and implementation container]

fm_shell (verify)> report_hdlin_mismatches -s -c myContainer
[Summary report of the named container]
```

report_hierarchy

Reports the hierarchy of a design.

SYNTAX

```
status report_hierarchy
   [designID]
   [-level integer]
```

Data Types

```
designID string
integer integer
```

ARGUMENTS

designID

Specifies a design. By default, the command reports the hierarchy of the current design.

-level *integer*

Report only the designs at or above this level (top level design is level 0). The specified integer must be greater than or equal to zero.

DESCRIPTION

This command reports the hierarchy of a design. The report does not include the hierarchy of a technology library component.

To report the hierarchy of a specific design, use the *designID* argument. If you do not specify a design, the tool reports the current design by default.

By default, the entire design hierarchy is displayed. By using the **-level** option, you can print up-to a level. Note that the numbering starts from 0 for the top level design and continues down the hierarchy.

Each report has a banner that includes the design name, optional level specification, followed by the number of unresolved references in the design.

EXAMPLES

This example shows part of the report for a large design:

```
fm_shell> report_hierarchy i:/bentle.db
**************************************************
Report         : hierarchy

Reference      : r:/bentle
Implementation : i:/bentle
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
******************************
Report : hierarchy
Design : bentle
******************************
Warning: 50 unresolved references are not included in
this report. (RPT-2)
bentle
FD1AH        cg51_convex
ITCUH        cg51_io_convex
IntCtrl
.
.
.
IntCtrl_qo
AF1L (24)    cg51_convex
AN4L cg51_convex
AN223L
AN224L  (unlinked)
fm_shell>
```

This example shows the usage of -level option:

```
fm_shell> report_hierarchy -level 3
**************************************************
Report         : hierarchy
                 -level 3

Reference      : r:/WORK/top
Implementation : <None>
Version        : L-2016.03
Date           : Tue Dec  8 02:23:08 2015
**************************************************


Design: top

top                                    WORK
    Formality Primitives (2)
    midA                               WORK
        Formality Primitives (2)
        botA                           WORK
```

```
            Formality Primitives (2)
            botB                       WORK
    midB                               WORK
        Formality Primitives (2)
        botB                           WORK
            Formality Primitives (2)
            dff                        WORK
1

fm_shell> report_hierarchy -level 1
**************************************************
Report         : hierarchy
                 -level 1

Reference      : r:/WORK/top
Implementation : <None>
Version        : L-2016.03
Date           : Tue Dec  8 02:23:08 2015
**************************************************


Design: top

top                                    WORK
    Formality Primitives (2)
    midA                               WORK
    midB                               WORK
1
```

SEE ALSO

report_designs(2)

report_host_options

Reports the options that set using the **set_host_options** command.

SYNTAX

```
status report_host_options
```

DESCRIPTION

This command reports the options that are set using the **set_host_options** command.

EXAMPLES

This example shows how to use the **report_host_options** command.

```
fm_shell (setup)> report_host_options
max_cores: 4
1
```

SEE ALSO

```
set_host_options(2)
```

report_init_toggle_assumption

Reports the initial toggle assumption on a controlling object(s).

SYNTAX

```
report_init_toggle_assumption
   [-all | object_list]
```

Data Types

```
object_list string
```

Enabled Shell Modes

All modes

ARGUMENTS

object_list

Specifies the object(s) whose initial toggle assumptions need to be reported.

-all

Reports all the initial toggle assumptions.

DESCRIPTION

Reports the initial toggle assumption on a controlling object(s) set by the set_init_toggle_assumption command.

EXAMPLES

The following example reports the initial toggle assumption on *i:/WORK/bit_slice/hclk*.

```
fm_shell (setup)> report_init_toggle_assumption i:/WORK/bit_slice/hclk
Object i:/WORK/bit_slice/hclk has been assumed to not toggle during initialization
1
```

The following example reports all the initial toggle assumptions..

```
fm_shell (setup)> report_init_toggle_assumption -all
Object i:/WORK/bit_slice/hclk has been assumed to not toggle during initialization
Object i:/WORK/bit_slice/lclk has been assumed to not toggle during initialization
1
```

SEE ALSO

```
set_init_toggle_assumption(2)
remove_init_toggle_assumption(2)
```

report_init_toggle_objects

Reports all objects that are assumed to toggle and not toggle that can initialize potentially constant registers. Includes objects identified by the initialization mode and those set by user.

SYNTAX

```
report_init_toggle_objects
   [-toggle_objects]
   [-no_toggle_objects]
```

ARGUMENTS

-toggle_objectsP

Report only the objects assumed to toggle that initialize potentially constant registers

-no_toggle_objects

Report only the objects assumed to not toggle that can initialize potentially constant registers

Enabled Shell Modes

Match
Verify

DESCRIPTION

This command reports all objects assumed to toggle and not toggle that can initialize potentially constant registers. Includes objects identified by the initialization mode and those set by user.

EXAMPLES

The following example reports all objects assumed to toggle and not toggle.

```
fm_shell (verify)> **report_init_toggle_objects**
**************************************************
Report         : init_toggle_objects

Reference      : r:/WORK/bit_slice
Implementation : i:/WORK/bit_slice
Version        : O-2018.06-BETA-20180520
Date           : Mon May 21 07:13:16 2018
**************************************************
No-Toggle Objects:
       -i:/WORK/bit_slice/badclk
1
```

SEE ALSO

```
report_potentially_constant_registers(2)
set_init_toggle_assumption(2)
report_init_toggle_assumption(2)
remove_init_toggle_assumption(2)
```

report_input_value_range

Reports the input value ranges set by the **set_input_value_range** command.

SYNTAX

```
status report_input_value_range
   [objectID_list]
```

Data Types

objectID_list string

Enabled Shell Modes

Setup

ARGUMENTS

objectID_list

Reports the input value range of the specified design objects.

DESCRIPTION

This command reports the input value ranges set by the **set_input_value_range** command.

EXAMPLES

The following example report input value range for all design objects.

```
fm_shell (setup)> report_input_value_range
**************************************************
Report         : input_value_range

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
Input       Object
Range       Type     Object Name
--------    ------    -----------
01X          port     r:/WORK/top/d
1X           port     r:/WORK/top/rst
1

The following example report input value range for the r:/WORK/top/PI1 object.

fm_shell (setup)> report_input_value_range r:/WORK/top/PI1
**************************************************
Report         : input_value_range

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
Input       Object
Range       Type     Object Name
--------    ------    -----------
0X           port     r:/WORK/top/PI1
1
```

SEE ALSO

```
set_input_value_range(2)
remove_input_value_range(2)
write_hierarchical_verification_script(2)
```

report_inv_push

Reports the inversions moved across register boundaries that are specified by the **set_inv_push** command, or via SVF inversion push guidance.

SYNTAX

```
status report_inv_push
   [designID]
   [-svf]
   [-diff]
   [-script]
```

Data Types

```
designID string
```

ARGUMENTS

designID

Reports the inversions moved across register boundaries in the specified design.

-svf

Reports the inversions moved across register boundaries via SVF inversion push guidance.

-diff

Implies -svf; reports the differences in SVF inversion push guidance between reference and implementation designs.

-script

Sames as -diff but the differences are reported in script format.

DESCRIPTION

This command reports the inversions moved across register boundaries that are specified by the **set_inv_push** command or via SVF inversion push guidance.

To list inversion push points that are defined for a specific design, specify the *designID* argument. If you do not specify the *designID* argument, the command reports all user- and SVF- defined inversion push points.

To list inversion push specified only via SVF inversion push guidance, specify the *-svf* argument. To limit the report to inversion push differences between reference and implementation, specify the *-diff* argument. If you specify the *-script* argument, the differences will be reported as a series of "set_user_match -inverted" commands.

EXAMPLES

This example sets a inversion push point and then reports it.

```
fm_shell> report_inv_push ref:/WORK/CORE
**************************************************
Report         : input_value_range

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

User Defined Inversion Push:
Object
 Type     Object Name
------    -----------
 cell     ref:/WORK/CORE/reg_A

SVF Defined Inversion Push:    None

1
```

SEE ALSO

```
set_inv_push(2)
remove_inv_push(2)
```

report_inversion

Reports information about the user-inserted inversions created using the **insert_inversion** command.

SYNTAX

```
status report_inversion
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **report_inversion** command reports information about inversion inserted using the **insert_inversion** command. The commands reports the total number of user-inserted inversion followed by the list of inversion objects.

EXAMPLES

The following example shows how to use the **report_inversion** command.

```
fm_shell> report_inversion
User Inserted Inversion:   2

   i:/WORK/top/out2_FM_INV
   i:/WORK/top/i1_FM_INV
1
```

SEE ALSO

```
remove_inversion(2)
insert_inversion(2)
```

report_libraries

Reports information about the current technology libraries.

SYNTAX

```
status report_libraries
   [-short]
   [-defects errors | all]
   [-merged_sequential_cells]
   [-unused_cells]
   [libraryID_list]
```

Data Types

```
libraryID_list string
```

ARGUMENTS

-short

Reports a summary of the technology libraries. Short, or summary, reports contain just the number of cells, the library name, and indicates whether the library is shared or not.

-defects errors | all

Reports information about defects or errors in the technology libraries.

-merged_sequential_cells

Reports information about merged registers in the technology libraries.

-unused_cells

Reports information about unused cells in the technology libraries. This option can not be used with '-defects errors' and '-merged_sequential_cells'.

library_list

Reports information about the specified technology libraries. Specify one of the following values for the *library_list* argument:

- *library_name* - Specifies the name of a shared technology library. Do not use a preceding slash character (/) as used when specifying a library ID.
- *libraryID* - Specifies a technology library in a specific container. You can use the wildcard character (*) as part of the design library name.

DESCRIPTION

This command reports information about technology libraries.

Detailed reports include a legend that helps in identifying the type of cell (sequential or black box). In these reports, the list of cells appears after the legend. For each library, the report identifies whether the library is shared or not. For technology libraries that are not shared, the report header includes the container name.

EXAMPLES

This example creates a short report about all technology libraries in the Formality environment. For each library, the report lists cells and their attributes.

```
fm_shell> report_libraries -short

   Number of
   Cells      Shared    Technology Library
   ---------  ------    ------------------
     79       Yes       gtech
     79       Yes       impl:/gtech
     79       Yes       ref:/gtech
1
fm_shell>
```

This example generates a long report about the shared technology library *gtech*. For brevity, the example omits most cell listings. Long reports have an attribute legend at the beginning of the report.

```
fm_shell> report_libraries gtech
**************************************************
Report         : libraries

Reference      : r:/TECH_WORK
Implementation : i:/TECH_WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
____________________________________________________________________________
|                                                                            |
|  Legend:                                                                   |
|           Attributes                                                        |
|             s = sequential cell                                             |
|             t = tristate                                                    |
|             b = BLACKBOX                                                    |
|_____________________________________________________________________________|
############################################################################
    SHARED TECH LIB - gtech
############################################################################
Library Cell                          Attributes
------------                          ----------
GTECH_ZERO
GTECH_ONE
GTECH_BUF
.
.
.
GTECH_ADD_ABC
GTECH_OUTBUF                            t
GTECH_INOUTBUF                          t
GTECH_INBUF
GTECH_TBUF                              t
GTECH_FD1                               b
GTECH_FD14                              s
GTECH_FD18                              s
.
.
.
GTECH_LD3                               b
GTECH_LD4                               b
GTECH_LSR0                              b
1
fm_shell>
```

This example creates a report about all the merged registers in technology libraries in the Formality environment.

```
fm_shell>report_libraries -merged_sequential_cells
**************************************************
Report         : libraries

Reference      : r:/TECH_WORK
Implementation : i:/TECH_WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
###############################################################################################
####    TECH LIB - r:/TECH_WORK
###############################################################################################
Library Cell                   Merged Register      Original Cells               Q         QN
------------                   ---------------      --------------              ---       ---
Simple_Cell                    *dff.00.Q*           UDP_DFFA, UDP_DFFB           Q         QX

1
```

This example generates a short report about unused cells in technology libraries in the Formality environment.

```
fm_shell>report_libraries -unused -short
**************************************************
Report         : libraries
                 -short
                 -unused_cells

Reference      : r:/WORK/top
```

```
Implementation : <None>
Version        : L-2016.03
Date           : Tue Jan  5 20:24:08 2016
**************************************************

  Unused
  Cells      Shared    Technology Library
---------    --------  ------------------
     89       Yes      GTECH
      1       Yes      TECH_WORK

     89       Yes      i:/GTECH
      1       Yes      i:/TECH_WORK

     89       Yes      r:/GTECH

1
```

SEE ALSO

```
list_libraries(2)
```

report_loops

Reports the nets and pins of loops or portions of loops.

SYNTAX

```
int report_loops
   [-ref | -impl]
   [-limit n]
   [-unfold]
```

Data Types

```
n integer
```

Enabled Shell Modes

Match Verify

ARGUMENTS

-ref

Reports information about the nets and pins of loops in the reference design. By default, the command reports information about loops in both the reference and implementation designs.

-impl

Reports information about the net and pins of loops in the implementation design. By default, the command reports information about loops in both the reference and implementation designs.

-limit *n*

Limits the report to the number of specified loops. By default, the tool reports 10 loops per design and 100 objects per loop. If you set the limit to 0, the tool reports all nets and pins in all loops of the specified design.

-unfold

Reports information about loops in overlapping (or embedded) loops as separate loops. By default, subloops are not reported. The report size might increase exponentially because of the number of subloops.

DESCRIPTION

This command reports information about the nets and pins of loops in the reference and implementation design.

Use this command either before or after the **verify** command. This command does not require verification results. The command reports all loops even if the loops are broken during verification.

EXAMPLES

The following example shows a report that does not contain subloops. The reported objects are instance path names.

```
prompt> report_loops -impl
**************************************************
Report         : loops

Reference      : SPEC:/WORK
Implementation : i:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
There is(are) 1 loop(s) found in the implementation design :

Loop 1 :
  (Pin) IMPL:/LIB/testImpl/D0/OUT
  (Net) IMPL:/LIB/testImpl/net_x1
  (Pin) IMPL:/LIB/testImpl/C0/IN1
  (Pin) IMPL:/LIB/testImpl/C0/OUT
  (Net) IMPL:/LIB/testImpl/net_b
  (Pin) IMPL:/LIB/testImpl/D0/F
1
```

The following example shows a report that contains subloops, without specifying the **-unfold** option.

```
prompt> report_loops -ref
**************************************************
Report         : libraries

Reference      : SPEC:/WORK
Implementation : i:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
```

```
There is(are) 1 loop(s) found in the reference design :

Loop 1 :
  (Pin) SPEC:/WORK/testSpec/C0/OUT
            ==> (Net) SPEC:/WORK/testSpec/w1
  (Net) SPEC:/WORK/testSpec/w1
            ==> (Pin) SPEC:/WORK/testSpec/C1/IN2
  (Pin) SPEC:/WORK/testSpec/C1/IN2
            ==> (Pin) SPEC:/WORK/testSpec/C1/OUT
  (Pin) SPEC:/WORK/testSpec/C1/OUT
            ==> (Net) SPEC:/WORK/testSpec/c
  (Net) SPEC:/WORK/testSpec/c
            ==> (Pin) SPEC:/WORK/testSpec/C0/IN2
            ==> (Pin) SPEC:/WORK/testSpec/C2/IN2
  (Pin) SPEC:/WORK/testSpec/C0/IN2
            ==> (Pin) SPEC:/WORK/testSpec/C0/OUT
  (Pin) SPEC:/WORK/testSpec/C2/IN2
            ==> (Pin) SPEC:/WORK/testSpec/C2/OUT
  (Pin) SPEC:/WORK/testSpec/C2/OUT
            ==> (Net) SPEC:/WORK/testSpec/w1
1
```

The following example shows a report using the **-unfold** option on a design with subloops.

```
prompt> report_loops -ref -unfold
**************************************************
Report         : libraries

Reference      : SPEC:/WORK
Implementation : i:/TECH_WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
There is(are) 1 loop(s) found in the reference design :

Loop 1 :

  (Sub-loop 1)
  (Pin) SPEC:/WORK/testSpec/C0/OUT
  (Net) SPEC:/WORK/testSpec/w1
  (Pin) SPEC:/WORK/testSpec/C1/IN2
  (Pin) SPEC:/WORK/testSpec/C1/OUT
  (Net) SPEC:/WORK/testSpec/c
  (Pin) SPEC:/WORK/testSpec/C0/IN2

  (Sub-loop 2)
  (Net) SPEC:/WORK/testSpec/w1
  (Pin) SPEC:/WORK/testSpec/C1/IN2
  (Pin) SPEC:/WORK/testSpec/C1/OUT
  (Net) SPEC:/WORK/testSpec/c
  (Pin) SPEC:/WORK/testSpec/C2/IN2
  (Pin) SPEC:/WORK/testSpec/C2/OUT
1
```

SEE ALSO

```
remove_cutpoint(2)
report_cutpoints(2)
```

set_cutpoint(2)

report_matched_points

Reports information about the matched compare points.

SYNTAX

```
integer report_matched_points
   [-compare_rule]
   [-substring substring]
   [-point_type point_type]
   [-inputs input_type]
   [-status status]
   [-except_status except_status]
   [-method matching_method]
   [-last]
   [-type ID_type]
   [-datapath]
   [-not_compared]
   [-list]
   [-never_loads]
   [-always_reset]
   [objectID]
```

Data Types

```
substring string
point_type string
input_type string
status string
except_status string
matching_method string
ID_type string
objectID string
```

Enabled Shell Modes

Match Verify

ARGUMENTS

-compare_rule

Reports mapped object names resulting from the application of compare rules.

-datapath

Reports only matched datapath blocks.

-substring *substring*

Reports only the points that contain the specified substring.

-point_type *point_type*

Reports only the points that match the specified type or types. By default, matchable input points and matchable compare points are reported. All point types are included in the report except blocks, block pins, and PGPin.

Specify one of the following values for the *point_type* argument:

- *all* - Report all object types.
- *bbox* - Report black boxes.
- *bbox_pin* - Report all black box pins.
- *bbox_input* - Report black box input pins.
- *bbox_output* - Report black box output pins.
- *bbox_inout* - Report black box bidirectional pins.
- *block* - Report hierarchical blocks.
- *block_pin* - Report hierarchical block pins.
- *block_input* - Report hierarchical block input pins.
- *block_output* - Report hierarchical block output pins.
- *block_inout* - Report hierarchical block bidirectional pins.
- *loop* - Report loop points.
- *net* - Report net points.
- *cut* - Report cutpoints.
- *bbox_net* - Report black box-resolved, multiply-driven nets.
- *port* - Report all ports.
- *input* - Report input ports.
- *output* - Report output ports.
- *inout* - Report bidirectional ports.
- *directly_undriven_output* - Report output and bidirectional ports that do not have connected nets or that have connected nets but do not have a driving pin.
- *reg* - Report all registers.
- *DFF* - Report flip-flop registers.

- *LAT* - Report latch registers.
- *PDCut* - Report power domain pin compare points.
- *PGPin* - Report power/ground/power-down function compare points.

-status *status*

Reports only the compare points that match the specified type or types. If both **-status** and **-point_type** arguments are specified, the compare points that are included must pass both filters.

By default, if neither the **-status** nor the **-except_status** options are used, the report includes points with any status except:

- Points where one or both objects are *unread*
- *dont_verify* points with a matched status that is not necessary for subsequent verification success

To exclude all *dont_verify* points from the report, use the **-except_status** *dont_verify* option. Multiple occurrences of this option are legal and values are additive. Specify one of the following values for the *status* argument:

- *none* - Report all except these special types.
- *trans* - Report transparent latch registers.
- *const* - Report all constant registers.
- *const0* - Report constant 0 registers.
- *const1* - Report constant 1 registers.
- *constX* - Report constant X registers.
- *const0X* - Report constrained 0X registers.
- *const1X* - Report constrained 1X registers.
- *clock_gate* - Report clock-gate latches.
- *dont_verify* - Report *dont_verify* points.
- *unread* - Report unread points.
- *undriven* - Report undriven points.
- *inverted* - Report points matched inverted.
- *noninverted* - Report points matched noninverted.
- *targeted* - Report points targeted for verification (requires the "Formality-Ultra" license key).
- *not_targeted* - Report points not targeted for verification (requires the "Formality-Ultra" license key).

-except_status *except_status*

Excludes points that match the specified status from the report. Multiple occurrences of this option are legal and values are additive. By default, the report includes dont_verify points with a matched status that could be necessary for downstream verification success. Specify one of the following values for the *except_status* argument:

- *trans* - Exclude transparent latch registers.
- *const* - Exclude all constant registers.
- *const0* - Exclude constant 0 registers.
- *const1* - Exclude constant 1 registers.
- *constX* - Exclude constant X registers.
- *const0X* - Exclude constrained 0X registers.
- *const1X* - Exclude constrained 1X registers.
- *clock_gate* - Exclude clock-gate latches.
- *dont_verify* - Exclude *dont_verify* points.
- *unread* - Exclude unread points.
- *undriven* - Exclude undriven points.
- *inverted* - Exclude points matched inverted.
- *noninverted* - Exclude points matched noninverted.
- *targeted* - Exclude points targeted for verification (requires the "Formality-Ultra" license key).
- *not_targeted* - Exclude points not targeted for verification (requires the "Formality-Ultra" license key).

-method *matching_method*

Reports only the points that match the specified method.

Specify one of the following values for the *matching_method* argument:

- *user* - Report points matched by the **set_user_match** command.
- *name* - Report points matched by name.
- *topology* - Report points matched by topology.
- *function* - Report points matched by function.

-last

Includes points matched during the most recent **match** or **verify** command.

-not_compared

Reports only points that are reported as "Not Compared" in the verification results summary.

-list

Returns a list of pairs of point names which can be used in further Tcl processing. By default, the command prints a formatted table.

-never_loads

Filters the report to only include register compare point pairs that do not synchronously load.

-always_reset

Filters the report to only include register compare point pairs that are constantly reset.

objectID* -type *ID_type

Reports points (matched cone inputs) in the compare point fanin specified by the *objectID* argument. Use the **-type** *ID_type* option to specify the type of the compare point if its name is ambiguous. These arguments are optional, but the **-type** *ID_type* option is only valid when used with the *objectID* argument.

DESCRIPTION

The **report_matched_points** command reports on design objects that are matched.

The tool generates a report that includes the total number of matched points followed by each pair of matched design objects. For each design object, the report lists the matching method, whether the match occurred in the most recent matching operation, the container, the design object type, and the design object's design ID.

EXAMPLES

The following command reports on all of the matched design objects:

```
prompt> report_matched_points
**************************************************
Report         : matched_points

Reference      : HDL:/WORK/CORE
Implementation : OPT:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

12 Matched points (substring 'INT_UPC_reg'):

  Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[10]
  Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[10]

  Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[11]
  Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[11]

  Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[12]
```

```
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[12]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[1]
 Impl DFF    (-) Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[1]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[2]
 Impl DFF    (-) Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[2]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[3]
 Impl DFF    (-) Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[3]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[4]
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[4]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[5]
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[5]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[6]
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[6]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[7]
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[7]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[8]
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[8]

 Ref DFF        Name(Last) HDL:/WORK/CORE/UPC_BLK/INT_UPC_reg[9]
 Impl DFF        Name(Last) OPT:/WORK/CORE/UPC_BLK/INT_UPC_reg[9]


[BBNet: multiply-driven net
 BBox:  black box
 BBPin: black box pin
 Block: hierarchical block
 BlPin: hierarchical block pin
 Cut:   cut-point
 DFF:   non-constant DFF register
 DFF0:  constant 0 DFF register
 DFF1:  constant 1 DFF register
 DFFX:  constant X DFF register
 DFF0X: constrained 0X DFF register
 DFF1X: constrained 1X DFF register
 LAT:   non-constant latch register
 LAT0:  constant 0 latch register
 LAT1:  constant 1 latch register
 LATX:  constant X latch register
 LAT0X: constrained 0X latch register
 LAT1X: constrained 1X latch register
 LATCG: clock-gating latch register
 TLA:   transparent latch register
 TLA0X: transparent constrained 0X latch register
 TLA1X: transparent constrained 1X latch register
 Loop:  cycle break point
 Net:   matchable net
 Port:  primary (top-level) port
 Und:   undriven signal cut-point

 Func:  matched by function
 Name:  matched by name
 Topo:  matched by topology
 User:  matched by user

 Last:  matched during most recent matching]
```

```
      1
```

SEE ALSO

```
report_aborted_points(2)
report_failing_points(2)
report_passing_points(2)
report_unmatched_points(2)
report_unverified_points(2)
report_not_compared_points(2)
```

report_mismatch_message_filters

Reports all the simulation-synthesis mismatch filters currently set on one or more simulation-synthesis mismatch messages.

SYNTAX

```
status report_mismatch_message_filters
   [-unmatched]
   [MismatchMessageIDList]
```

Data Types

MismatchMessageIDList	list

ARGUMENTS

-unmatched

Reports the mismatch filters those are not executed by any of the mismatch-messages produced during the run. This is an optional option. By default, the report command will display all the filters.

MismatchMessageIDList

Reports mismatch message filter for the list of mismatch message Ids specified. This is an optional option. Below are the list of of simulation mismatch error codes that this option accepts:

FMR_VHDL-274 FMR_VHDL-1002 FMR_VHDL-1004 FMR_VHDL-1014 FMR_VHDL-1025 FMR_VHDL-1027 FMR_VHDL-1036 FMR_VHDL-1140 FMR_VHDL-1144 FMR_VHDL-1145 FMR_VLOG-079 FMR_VLOG-081 FMR_VLOG-083 FMR_VLOG-087 FMR_VLOG-089 FMR_VLOG-090 FMR_VLOG-091 FMR_VLOG-925 FMR_VLOG-928 FMR_VLOG-929 FMR_ELAB-034 FMR_ELAB-058 FMR_ELAB-059 FMR_ELAB-100 FMR_ELAB-115 FMR_ELAB-116 FMR_ELAB-117 FMR_ELAB-118 FMR_ELAB-125 FMR_ELAB-130 FMR_ELAB-136 FMR_ELAB-145 FMR_ELAB-146 FMR_ELAB-147 FMR_ELAB-149 FMR_ELAB-150 FMR_ELAB-151 FMR_ELAB-153 FMR_ELAB-154 FMR_ELAB-261

DESCRIPTION

The command reports all the filters set in the order of priority. The fist displayed filter for a specific message will have highest priority than the next in the report.

The command returns status 1 on success and 0 on failure.

EXAMPLES

Assume below filters set in a session:

```
set_mismatch_message_filter -warn -block bot_sizeA1_size0/gen0/alwaysb FMR_ELAB-117
set_mismatch_message_filter -suppress -file /vobs/data/rtl/test.sv -line 57 FMR_ELAB-117
set_mismatch_message_filter -warn
```

Then the report command reports the filters as below

```
fm_shell (setup)> report_mismatch_message_filters
**************************************************
Report         : report_mismatch_message_filters

Reference      : <None>
Implementation : <None>
Version        : J-2014.09
Date           : Wed Jul 15 01:58:10 2014
**************************************************

   FMR_ELAB-117  -suppress -file /vobs/data/rtl/test.sv -line 57
   FMR_ELAB-117      -warn -block bot_sizeA1_size0/gen0/alwaysblock
   FMR_ELAB-117      -warn

                     -warn
```

In above report, the first three filters for mismatch message ID FMR_ELAB-117 and the last filter is for all other mismatch message IDs. The third line corresponds to implicit filter that set on FMR_ELAB-117 because of set_mismatch_message_filter -warn.

If there are many filters set, and you want to report the filter set on particular message ID, then use the below shown command:

```
fm_shell (setup)> report_mismatch_message_filters FMR_ELAB-117
**************************************************
Report         : report_mismatch_message_filters
                   FMR_ELAB-117
Reference      : <None>
Implementation : <None>
Version        : J-2014.09
Date           : Wed Jul 16 03:21:55 2014
**************************************************

   FMR_ELAB-117  -suppress -file /vobs/data/rtl/test.sv -line 57
   FMR_ELAB-117  -suppress -signal data1
   FMR_ELAB-117      -warn -block bot_sizeA1_size0/gen0/alwaysblock
   FMR_ELAB-117  -suppress
```

If you want to check the filters that are not matched by any of the mismatch message, use the command as shown below:

```
fm_shell (setup)> report_mismatch_message_filters FMR_ELAB-117 -unmatched
**************************************************
Report         : report_mismatch_message_filters
                   FMR_ELAB-117
                 -unmatched
Reference      : r:/WORK/small_test
Implementation : <None>
Version        : J-2014.09-Beta2
Date           : Wed Jul 16 03:32:30 2014
**************************************************

   FMR_ELAB-117      -warn -block bot_sizeA1_size0/gen0/alwaysblock
   FMR_ELAB-117  -suppress
```

Please note that, **-unmatched** will report those filters that are not hit by any of the mismatch messages. It does not display implicit filter(a filter set without MismatchMessageIDList) on specific message ID like normal report.

SEE ALSO

```
set_mismatch_message_filter(2)
remove_mismatch_message_filter(2)
```

report_multidriven_nets

Reports information about multiply-driven nets after match.

SYNTAX

```
report_multidriven_nets
   [-substring substring]
   [-reference]
   [-implementation]
```

Data Types

substring string

Enabled Shell Modes

Match
Verify

ARGUMENTS

-substring *substring*

Reports information about only the nets that contain the specified substring.

-reference

Reports information about only the multiply-driven nets in the reference design. By default, the command reports multiply-driven nets in both the reference and implementation designs.

-implementation

Reports information about only the multiply-driven nets in the implementation design. By default, the command reports multiply-driven nets in both the reference and implementation designs.

DESCRIPTION

This command reports information about the multiply-driven nets after match. The report includes the total number of multiply-driven nets, followed by each multiply-driven net and the list of drivers. The nets and drivers include the full instance path names. The resolution or wire type is shown for each net if it is not of consensus type. The cell and library names are displayed for each driver if they exist.

EXAMPLES

The following example shows a report generated by the **report_multidriven_nets** command.

```
prompt> report_multidriven_nets
***************************************************
Report         : multidriven_nets

Reference      : r:/WORK
Implementation : i:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
***************************************************

report_multidriven_nets

2 Multiply driven nets:

Net    r:/WORK/top/Z
Drivers (3)
       r:/WORK/and_multi/C1/OUT
       r:/WORK/top/mid/out      (inv)   (TECH_WORK)
       r:/WORK/and_multi/C0/OUT

Net    i:/WORK/top/Z
Drivers (3)
       i:/WORK/and_multi/C1/OUT
       i:/WORK/top/mid/out      (inv)   (TECH_WORK)
       i:/WORK/and_multi/C0/OUT

1
```

SEE ALSO

```
report_undriven_nets(2)
```

report_net_resolution

Reports resolution function for the specified net.

SYNTAX

```
status report_net_resolution
   [ objectID ]
```

Data Types

objectID	string

Enabled Shell Modes

Setup

ARGUMENTS

objectID

Specifies the net for which the resolution function is to be reported. If you specify a name that resolves to more than one net, the resolution function is reported for all of the matching nets.

DESCRIPTION

Use this command to report the resolution function for a net. If a resolution function has not been specified for the net, the default resolution function will be reported.

If the command is invoked with no arguments, all the nets with user specified resolution function are reported.

EXAMPLES

The following examples reports the resolution function for the net VDD in the implementation design.

```
fm_shell (setup)> report_net_resolution \
i:/WORK/dut/VDD
**************************************************
Report         : net_resolution_functions

Reference      : r:/WORK/dut
Implementation : i:/WORK/dut
Version        : O-2018.06
Date           : Wed Feb 21 09:37:45 2018
**************************************************
Resolution Function  Net Name
===================  ========
parallel             i:/WORK/dut/VDD
1
```

SEE ALSO

```
remove_net_resolution(2)
set_net_resolution(2)
```

report_not_compared_points

Reports compare points that it was unnecessary to compare.

SYNTAX

```
int report_not_compared_points
   [-compare_rule]
   [-substring substring]
   [-point_type point_type]
   [-status status]
   [-except_status except_status]
   [-list]
```

Data Types

```
substring string
point_type string
status string
except_status string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-compare_rule

Reports mapped object names resulting from the application of compare rules.

-substring *substring*

Reports only the points containing the specified *substring*.

-point_type *point_type*

Filters the report to include only the compare points with characteristics that match the specified type or types. By default, all compare point types are included. Specify one or more *point_type* values on which to report.

The valid *point_type* values are as follows:

- **bbox_pin** - Report all black-box pin compare points.
- **bbox_input** - Report black-box input pin compare points.
- **bbox_inout** - Report black-box bidirectional pin compare points.
- **loop** - Report loop compare points.
- **bbox_net** - Report black-box-resolved multiply-driven nets.
- **cut** - Report cut-points.
- **port** - Report all port compare points.
- **output** - Report output port compare points.
- **inout** - Report bidirectional port compare points.
- **directly_undriven_output** - Report output and bidirectional ports that do not have a connected net, or that have a connected net but do not have a driving pin.
- **reg** - Report all register compare points.
- **DFF** - Report flip-flop register compare points.
- **LAT** - Report latch register compare points.
- **trans** - Report transparent latch register compare points.
- **PDCut** - Report power domain pin compare points.
- **PGPin** - Report power/ground/power-down function compare points.

-status ***status***

Reports only the points that match the specified type or types. If both **-status** and **-point_type** are specified, included points must pass both filters.

The valid *status* values are as follows:

- **none** - Report all except these special types.

- **trans** - Report transparent latch registers.
- **const** - Report all constant registers.
- **const0** - Report constant 0 registers.
- **const1** - Report constant 1 registers.
- **constX** - Report constant X registers.
- **const0X** - Report constrained 0X registers.
- **const1X** - Report constrained 1X registers.
- **clock_gate** - Report clock-gate latches.

- **dont_verify** - Report *dont_verify* points.
- **unread** - Report unread points.
- **undriven** - Report undriven points.
- **inverted** - Report points matched inverted.
- **noninverted** - Report points matched noninverted.
- *targeted* - Report points targeted for verification (requires the "Formality-Ultra" license key).
- *not_targeted* - Report points not targeted for verification (requires the "Formality-Ultra" license key).

-except_status *except_status*

Filters the report to exclude points that match the specified type. Multiple occurrences of this switch are legal and values are additive. The valid *except_status* values are as follows:

- `trans - Exclude transparent latch registers.`

- **const** - Exclude all constant registers.
- **const0** - Exclude constant 0 registers.
- **const1** - Exclude constant 1 registers.
- **constX** - Exclude constant X registers.
- **const0X** - Exclude constrained 0X registers.
- **const1X** - Exclude constrained 1X registers.
- **clock_gate** - Exclude clock-gate latches.
- **dont_verify** - Exclude *dont_verify* points.
- **unread** - Exclude unread points.
- **undriven** - Exclude undriven points.
- **inverted** - Exclude points matched inverted.
- **noninverted** - Exclude points matched noninverted.
- *targeted* - Exclude points targeted for verification (requires the "Formality-Ultra" license key).
- *not_targeted* - Exclude points not targeted for verification (requires the "Formality-Ultra" license key).

-list

Reports the not compared points as a list. You can use this list for further Tcl processing.

DESCRIPTION

The **report_not_compared_points** command reports points that it was not necessary to compare because they were either unread, constant, marked dont_verify, or clock-gating latches.

Formality produces a report that includes the total number of points reported followed by each pair of matched design objects. For each design object, the report lists the matching method, whether the match occurred in the most recent matching operation, the container, the design object type, and the design object's *designID*.

EXAMPLES

The following example reports on all of the not compared design objects:

```
prompt> report_not_compared_points

27 Not compared points:

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/emif0/emif_clocks0/clkgate_inp_sb
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/emif0/emif_clocks0/clkgate_inp_sb

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/emif0/emif_clocks0/clkgate_inp_sd
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/emif0/emif_clocks0/clkgate_inp_sd

  Ref  LAT        Name(Last) ref:/LAPLACE_LIB/laplace/emif0/emif_clocks0/clkgate_inp_tck
  Impl LAT        Name(Last) der:/LAPLACE_LIB/laplace/emif0/emif_clocks0/clkgate_inp_tck

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/emif0/emif_clocks0/reset_cntr_reg/cg0
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/emif0/emif_clocks0/reset_cntr_reg/cg0

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/emif1/emif_clocks0/clkgate_inp_sb
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/emif1/emif_clocks0/clkgate_inp_sb

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/emif1/emif_clocks0/clkgate_inp_sd
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/emif1/emif_clocks0/clkgate_inp_sd

  Ref  LAT        Name(Last) ref:/LAPLACE_LIB/laplace/emif1/emif_clocks0/clkgate_inp_tck
  Impl LAT        Name(Last) der:/LAPLACE_LIB/laplace/emif1/emif_clocks0/clkgate_inp_tck

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/emif1/emif_clocks0/reset_cntr_reg/cg0
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/emif1/emif_clocks0/reset_cntr_reg/cg0

  Ref  LATCG      Name(Last) ref:/LAPLACE_LIB/laplace/padf0/pfmisc0/bea_reg0/cg0
  Impl LATCG      Name(Last) der:/LAPLACE_LIB/laplace/padf0/pfmisc0/bea_reg0/cg0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg0/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg0/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg1/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg1/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg2/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg2/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg3/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg3/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg4/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg4/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg5/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg5/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg6/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg6/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg7/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg7/u0

  Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg8/u0
  Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_0/nscanreg8/u0
```

```
 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg0/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg0/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg1/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg1/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg2/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg2/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg3/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg3/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg4/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg4/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg5/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg5/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg6/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg6/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg7/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg7/u0

 Ref  DFFX       Name(Last) ref:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg8/u0
 Impl DFFX       Name(Last) der:/LAPLACE_LIB/laplace/test_str0/rring_osc_1/nscanreg8/u0

[BBNet: multiply-driven net
 BBox:  black-box
 BBPin: black-box pin
 Block: hierarchical block
 BlPin: hierarchical block pin
 Cut:   cut-point
 DFF:   non-constant DFF register
 DFF0:  constant 0 DFF register
 DFF1:  constant 1 DFF register
 DFFX:  constant X DFF register
 DFF0X: constrained 0X DFF register
 DFF1X: constrained 1X DFF register
 LAT:   non-constant latch register
 LAT0:  constant 0 latch register
 LAT1:  constant 1 latch register
 LATX:  constant X latch register
 LAT0X: constrained 0X latch register
 LAT1X: constrained 1X latch register
 LATCG: clock-gating latch register
 TLA:   transparent latch register
 TLA0X: transparent constrained 0X latch register
 TLA1X: transparent constrained 1X latch register
 Loop:  cycle break point
 Net:   matchable net
 Port:  primary (top-level) port
 Und:   undriven signal cut-point

 Func:  matched by function
 Name:  matched by name
 Topo:  matched by topology
 User:  matched by user

 Last:  matched during most recent matching]
```

SEE ALSO

```
report_aborted_points(2)
report_failing_points(2)
report_matched_points(2)
report_passing_points(2)
report_unverified_points(2)
```

report_parameters

Reports information about parameters that are set by the **set_parameters** command.

SYNTAX

```
status report_parameters
   [designID_list]
```

Data Types

```
designID_list string
```

ARGUMENTS

designID_list

Reports parameters in the specified designs. By default, the command reports information about parameters in the current design.

DESCRIPTION

This command reports information about the user-defined parameters for the specified designs or the current design.

The report includes the following design information:

- Number of cells (linked and unlinked)
- Number of ports
- Number of nets

- Net resolution type
- Attributes of the flatten parameter
- Attributes of the retimed parameter

EXAMPLES

This example creates a report about parameters in the implementation and reference designs. One design is in the *impl* container, and another is in the *ref* container. The designs are in the WORK design library.

```
fm_shell> report_parameters {ref:/WORK/CORE impl:/WORK/CORE}
**************************************************
Report         : parameters

Reference      : ref:/WORK/CORE
Implementation : impl:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
 ___________________________________________________________________________
|                                                                           |
|  Legend:                                                                  |
|         Statistics                  Attributes                            |
|           c = number of cells         c = CONSENSUS resolution type       |
|               linked(unlinked)        x = BLACKBOX resolution type        |
|           p = number of ports         a = AND resolution type             |
|           n = number of nets          o = OR resolution type              |
|                                      f = flatten                          |
|                                      r = retimed                          |
|___________________________________________________________________________|
###########################################################################
    DESIGN LIBRARY - impl:/WORK
###########################################################################
Design Name                              Statistics                  Attributes
-----------                              ----------                  ----------
CORE                                     c10(0) p39 n102                c
###########################################################################
####    DESIGN LIBRARY - ref:/WORK
###########################################################################
Design Name                              Statistics                  Attributes
-----------                              ----------                  ----------
CORE                                     c10(0) p38 n102                c
1
```

SEE ALSO

```
remove_parameters(2)
set_parameters(2)
```

report_passing_points

Reports information about compare points that passed verification.

SYNTAX

```
int report_passing_points
   [-compare_rule]
   [-substring substring]
   [-point_type point_type]
   [-status status]
   [-inputs input_type]
   [-list]
   [-last]
   [-never_loads]
   [-always_reset]
```

Data Types

```
substring string
point_type string
status string
input_type string
```

Enabled Shell Modes

Verify

ARGUMENTS

-compare_rule

Reports the mapped object names resulting from the application of compare rules.

-substring *substring*

Reports information about the points that contain the specific substring.

-point_type *point_type*

Reports information about the compare points with characteristics that match the specified types. By default, all compare point types except PGPin are included.

Specify one of the following values for the *point_type* argument:

- *bbox_pin* - to report all black box pin compare points.
- *bbox_input* - to report black box input pin compare points.
- *bbox_inout* - to report black box bidirectional pin compare points.
- *loop* - to report loop compare points.
- *bbox_net* - to report black box-resolved multiply-driven nets.
- *cut* - to report cutpoints.
- *port* - to report all port compare points.
- *output* - to report output port compare points.
- *inout* - to report bidirectional port compare points.
- *directly_undriven_output* - to report output and bidirectional ports that do not have a connected net, or that have a connected net but do not have a driving pin.
- *reg* - to report all register compare points.
- *DFF* - to report flip-flop register compare points.
- *LAT* - to report latch register compare points.
- *trans* - to report transparent latch register compare points.
- *PDCut* - to report power domain pin compare points.
- *PGPin* - to report power/ground/power-down function compare points.

-status *status*

Reports information about the compare points with the specified polarity. By default, compare points of both polarities are included in the report.

Specify one of the following values for the *status* argument:

- *inverted* - to report compare points with inverted match polarity.
- *noninverted* - to report compare points with noninverted match polarity.

-inputs *input_type*

Reports information about the compare points with input points that are either undriven or unmatched, or both. Specify one or both *input_type* values on which to report.

- *undriven* - to report compare points with undriven input points.
- *unmatched* - to report compare points with unmatched input points.

-list

Reports a list of compare point pairs. You can use this list in further Tcl processing.

-last

Reports information about compare point pairs from the previous verification.

-never_loads

Filters the report to only include register compare point pairs that do not synchronously load.

-always_reset

Filters the report to only include register compare point pairs that are constantly reset.

DESCRIPTION

This command reports information about the compare points that passed verification. The report includes the total number of passing compare points, followed by each pair of design objects that passed verification. Information for each design object includes the container in which the design is, the design object type, and the design object's design ID.

EXAMPLES

The following example shows how to use the **report_passing_points** command:

```
prompt> report_passing_points
**************************************************
Report         : passing_points

Reference      : OPT:/WORK/CORE
Implementation : SCAN:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

16 Passing compare points:

  Ref Port       OPT:/WORK/CORE/FULL
  Impl Port       SCAN:/WORK/CORE/FULL

  Ref Port       OPT:/WORK/CORE/PL
  Impl Port       SCAN:/WORK/CORE/PL

  Ref Port       OPT:/WORK/CORE/PROM
  Impl Port       SCAN:/WORK/CORE/PROM

  Ref Port       OPT:/WORK/CORE/VECT
  Impl Port       SCAN:/WORK/CORE/VECT
```

```
 Ref Port       OPT:/WORK/CORE/Y[10]
 Impl Port       SCAN:/WORK/CORE/Y[10]

 Ref Port       OPT:/WORK/CORE/Y[11]
 Impl Port       SCAN:/WORK/CORE/Y[11]

 Ref Port       OPT:/WORK/CORE/Y[12]
 Impl Port       SCAN:/WORK/CORE/Y[12]

 Ref Port       OPT:/WORK/CORE/Y[1]
 Impl Port       SCAN:/WORK/CORE/Y[1]

 Ref Port       OPT:/WORK/CORE/Y[2]
 Impl Port       SCAN:/WORK/CORE/Y[2]

 Ref Port       OPT:/WORK/CORE/Y[3]
 Impl Port       SCAN:/WORK/CORE/Y[3]

 Ref Port       OPT:/WORK/CORE/Y[4]
 Impl Port       SCAN:/WORK/CORE/Y[4]

 Ref Port       OPT:/WORK/CORE/Y[5]
 Impl Port       SCAN:/WORK/CORE/Y[5]

 Ref Port       OPT:/WORK/CORE/Y[6]
 Impl Port       SCAN:/WORK/CORE/Y[6]

 Ref Port       OPT:/WORK/CORE/Y[7]
 Impl Port       SCAN:/WORK/CORE/Y[7]

 Ref Port       OPT:/WORK/CORE/Y[8]
 Impl Port       SCAN:/WORK/CORE/Y[8]

 Ref Port       OPT:/WORK/CORE/Y[9]
 Impl Port       SCAN:/WORK/CORE/Y[9]

[BBNet: multiply-driven net
 BBPin: black box pin
 Cut:   cut-point
 DFF:   non-constant DFF register
 DFF0:  constant 0 DFF register
 DFF1:  constant 1 DFF register
 DFFX:  constant X DFF register
 DFF0X: constrained 0X DFF register
 DFF1X: constrained 1X DFF register
 LAT:   non-constant latch register
 LAT0:  constant 0 latch register
 LAT1:  constant 1 latch register
 LATX:  constant X latch register
 LAT0X: constrained 0X latch register
 LAT1X: constrained 1X latch register
 LATCG: clock-gating latch register
 TLA:   transparent latch register
 TLA0X: transparent constrained 0X latch register
 TLA1X: transparent constrained 1X latch register
 Loop:  cycle break point
 Port:  primary (top-level) port
 Und:   undriven signal cut-point]

1
```

SEE ALSO

```
report_aborted_points(2)
report_failing_points(2)
report_matched_points(2)
report_unmatched_points(2)
report_unverified_points(2)
report_not_compared_points(2)
```

report_potentially_constant_registers

Reports information on the potentially constant registers.

SYNTAX

```
report_potentially_constant_registers
   [-info]
   [-fanout_of ObjectList]
   [-assumed_no_toggle]
```

ARGUMENTS

-fanout_of *ObjectList*

Reports only potentially constant registers in the fanout of the specified object. If the objectID was not the cause of initialization of any registers, the report will be empty (not an error) the objectID_list can be a list {a b c } of objects that are the possible toggling objects

-assumed_no_toggle

Reports only the potentially constant registers that were not found constant because some signal was assumed to *not* toggle (due to automatched mode, or user specified set_init_toggle_assumption -no_toggle)

-info

Provides additional information on why the constant registers were found to be constant or not.

Enabled Shell Modes

Match
Verify

DESCRIPTION

This command reports information about constant registers that were found to be constant based on toggle signal(s), and potentially constant registers that were not found constant because some signal was assumed to *not* toggle.

EXAMPLES

```
The following example reports on all potentially constant registers.
fm_shell> report_potentially_constant_registers
**************************************************
Report         : potentially_constant_registers

Reference      : r:/WORK/bit_slice
Implementation : i:/WORK/bit_slice
Version        : O-2018.06-ALPHA-20180328
Date           : Wed Mar 28 10:56:55 2018
**************************************************
(LAT0)  i:/WORK/bit_slice/my_latch1
1

The following example gives additional info
fm_shell> report_potentially_constant_registers -info
**************************************************
Report         : potentially_constant_registers
                 -info

Reference      : r:/WORK/bit_slice
Implementation : i:/WORK/bit_slice
Version        : O-2018.06-ALPHA-20180328
Date           : Wed Mar 28 10:56:55 2018
**************************************************
(LAT0)  i:/WORK/bit_slice/my_latch1
          - Data pin (AD) is constrained to 0
          - Enable pin (AL) can controlled by i:/WORK/bit_slice/hclk
1

The following example reports on all potentially constant registers that can be initialized by an o
fm_shell> report_potentially_constant_registers -fanout_of i:/WORK/bit_slice/hclk
**************************************************
Report         : potentially_constant_registers
    -fanout_of i:/WORK/bit_slice/hclk
Reference      : r:/WORK/bit_slice
Implementation : i:/WORK/bit_slice
Version        : O-2018.06-ALPHA-20180328
Date           : Wed Mar 28 10:56:55 2018
**************************************************
(LAT0)  i:/WORK/bit_slice/my_latch1
1

The following example reports on all potentially constant registers whose control signals have been
fm_shell (verify)> report_potentially_constant_registers -assumed_no_toggle
**************************************************
Report         : potentially_constant_registers
                 -assumed_no_toggle

Reference      : r:/WORK/bit_slice
Implementation : i:/WORK/bit_slice
Version        : O-2018.06-BETA-20180520
```

```
Date           : Mon May 21 07:14:24 2018
**************************************************
(LAT0X)  i:/WORK/bit_slice/my_latch1
1
```

SEE ALSO

```
report_init_toggle_objects(2)
set_init_toggle_assumption(2)
report_init_toggle_assumption(2)
remove_init_toggle_assumption(2)
```

report_probe_points

Reports the probe net pairs set by the **set_probe_points** command.

SYNTAX

```
report_probe_points
```

Enabled Shell Modes

Setup
Match
Verify

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports the debug probe pairs set by the **set_probe_points** command. The probe pair consists of a reference net and an implementation net.

EXAMPLES

```
fm_shell (setup)> report_probe_points
**********************************************
Report         : probe_points

Reference      : r:/WORK/top
```

```
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

    r:/WORK/top/data
    i:/WORK/top/data

    r:/WORK/top/U2/Z[0]
    i:/WORK/top/U3/m3/net4

    r:/WORK/top/M1/x
    i:/WORK/top/x
1
```

SEE ALSO

```
set_probe_points(2)
remove_probe_points(2)
report_probe_status(2)
```

report_probe_status

Reports verification status information about probe net pairs that are set by the **set_probe_points** command.

SYNTAX

```
report_probe_status
   [-status pass | fail | abort | notrun]
```

Enabled Shell Modes

Verify

ARGUMENTS

-status

Reports information about probe net pairs with the specified status. Specify one of the following values:

- *pass*
- *fail*
- *abort*
- *notrun*

DESCRIPTION

This command reports information about the verification status of debug probe pairs that are set by the **set_probe_points** command.

EXAMPLES

This example shows a report with all the probe net pairs and their verification status.

```
fm_shell (setup)> report_probe_status
**************************************************
Report         : probe_status

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

  Ref FAIL       r:/WORK/top/data
  Impl FAIL       i:/WORK/top/data

  Ref PASS       r:/WORK/top/U1/U3/net4
  Impl PASS       i:/WORK/top/U4/net9

  Ref FAIL       r:/WORK/top/M1/M2//Z[0]
  Impl FAIL       i:/WORK/top/Z_0_
1
```

This example shows a report with the probe net pairs that have passed verification.

```
fm_shell (setup)> report_probe_status  -status pass
**************************************************
Report         : probe_status

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************
  Ref PASS       r:/WORK/top/U1/U3/net4
  Impl PASS      i:/WORK/top/U4/net9
1
```

SEE ALSO

```
set_probe_points(2)
remove_probe_points(2)
report_probe_points(2)
verify(2)
```

report_related_supplies

Reports the driver and receiver supply nets and those that exist for the specified port or net.

SYNTAX

```
report_related_supplies
   ObjectPath
   [-drivers]
   [-receivers]
   [-type]
```

Data Types

```
ObjectPath string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-drivers

Reports the UPF driver supply information of the specified port or net.

-receivers

Reports the UPF receiver supply information of the specified port or net.

-type *type*

Specifies the object type. Use this option if the name of the specified design object is associated with more than one object type in the design. Specify one of the following values for the *type* argument:

- *port*

- *net*

DESCRIPTION

This command reports the driver and receiver supply nets and supply sets of the specified port or net. If the specified regular expression resolves to more than one object, then all of those objects will be reported. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the following precedence: port, net. By default, the command reports the full path name of the specified object, the drivers and the receivers.

Use only either the **-drivers** or the **-receivers** option to report only drivers or receivers respectively.

EXAMPLES

The following example shows how to report the UPF related supply information.

```
fm_shell> report_related_supplies r:/WORK/simple/n4[3]

**************************************************
Report         : upf_related_supplies
                 r:/WORK/simple/n4[3]

Reference      : r:/WORK/simple
Implementation : i:/WORK/simple
Version        : G-2012.06-SP1-2
Date           : Thu Aug  2 14:52:23 2012
**************************************************

Net: r:/WORK/simple/n4[3]
  Driver:
    r:/WORK/simple/u4/out1[3]
      Supplies: r:/WORK/simple/VDD1  r:/WORK/simple/VSS1
      Supply Set(s): /PD_TOP.primary /PD_A.iso_A1.isolation_supply_set
  Receiver:
    r:/WORK/simple/in1[3]_UPF_ISO/IN
      Supplies: r:/WORK/simple/VDD3  r:/WORK/simple/VSS2
      Supply Set(s): /PD_B.primary /PD_B.iso_B1.isolation_supply_set
1
```

SEE ALSO

```
load_upf(2)
report_upf(2)
```

report_remove_objects

Reports the objects that are removed by using the **remove_object** command.

SYNTAX

```
status report_remove_objects
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports the objects removed by the user using the \fremove_object command.

EXAMPLES

```
fm_shell (setup)> report_remove_objects
**************************************************
Report         : remove_objects

Reference      : <None>
Implementation : <None>
Version        : J-2014.09-Alpha
Date           : Fri May 16 12:01:16 2014
**************************************************

Removed Objects:

    (Port)  impl:/mapped_gate_IBM_CMOS5S_GApower.db/CORE/PL
```

```
        (Block) impl:/mapped_gate_IBM_CMOS5S_GApower.db/CORE/CNTL_BLK
        (Net)   impl:/mapped_gate_IBM_CMOS5S_GApower.db/MUX_OUT/DATA_1
        (Block) impl:/mapped_gate_IBM_CMOS5S_GApower.db/MUX_OUT/MUX_BLK

    1
```

SEE ALSO

```
remove_object(2)
```

report_setup_status

Reports information about the design statistics and warning messages that are issued by the tool during design read and user-specified setup.

SYNTAX

```
status report_setup_status
   [-design_info]
   [-hdl_read_messages]
   [-commands]
```

ARGUMENTS

-design_info

Reports information about the design.

-hdl_read_messages

Reports information about the warning messages that are issued by the tool when the design is read.

-commands

Reports user-specified setup commands.

DESCRIPTION

This command reports the design statistics and warning messages. This report enables you to check for and complete any missing design setup information before proceeding with the **match** and **verify** commands.

Any changes that are performed using the **synopsys_auto_setup** variable or the automated setup file for verification are not included in the report. Also, the reported statistics might differ from the statistics reported after **match** and **verify** commands.

The **report_setup_status** command can be run only after reading and linking both reference and implementation containers. The statistics reported in the user-specified setup section are the number of objects that a specific command addresses in a container.

EXAMPLES

This example shows a setup status report.

```
fm_shell (setup)> report_setup_status
**************************************************
Report         : setup_status

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

   #### Design Information ####
   # Design Settings #
      set_top reference design:  r:/WORK/top
      set_top implementation design:  i:/WORK/top
      set_reference_design:  r:/WORK/reference
      set_implementation_design:  i:/WORK/implementation

   # Design Statistics Ref (Imp) #
      Ports:            1200 (1203)
      Registers:        2300 (2351)
      Black boxes:         8 (8)
         - Unresolved modules: 1  (1)
         - User specified:     2  (2)
      Undriven nets:           41 (41)
      Multiply-driven nets:    5  (5)


   #### HDL Read Message Summary ####
      Message ID:      Occurrences:  Ref(Imp)
      ------------     -----------------------------
      FMR_VLOG-069                 1 (0)


   #### User Specified Setup ####
      Command Name:          Result:  Ref(Imp)
      --------------  -----------------------------
      set_black_box             :       3(  11)
      set_clock                 :       2(   1)
      set_compare_rule          :       2(   0)
      set_constant              :       4(   2)
      set_constraint            :       5(   5)
      set_cutpoint              :       0(   6)
      set_dont_cut              :       0(   0)
      set_dont_match_points     :       0(   0)
      set_dont_verify_point     :       8(   8)
      set_factor_point          :       0(   0)
      set_inv_push              :      11(  11)
```

```
        set_user_match            :      13 ( 13)
        set_verify_points         :       0(   0)
  1
```

SEE ALSO

report_source_path

Reports source path information of designs or instances.

SYNTAX

```
status report_source_path
   [-r]
   [-i]
   [-container container_name]
   [-tech]
   [ObjectID_list]
```

Data Types

```
ObjectID_list string/collection
```

ARGUMENTS

-r

Reports designs referenced in Reference container.

-i

Reports designs referenced in Implementation container.

-c container_name

Reports designs referenced in given container.

-tech

Reports objects in the technology libraries.

ObjectID_list

Reports information about the specified Designs or Instances. If the ObjectID is an instance, it reports source path of the design being referenced. ObjectOd_list is mutually exclusive to -i, -r and -c . ObjectID list can contain wild cards or a collection.

DESCRIPTION

This command reports source path information about one or more designs or instances.

Detailed reports include source file path, library name and container name of given design. If user has not specified any options, it reports the source path information about designs in all available containers.

EXAMPLES

This example creates a report about all technology libraries in the Formality environment.

```
fm_shell> report_source_path -tech
**************************************************
Report         : source_path
                 -tech

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : M-2016.12-SP3
Date           : Thu Mar 16 03:56:32 2017
**************************************************
#-----------------------------------------------------------
#       Container : i
#-----------------------------------------------------------

Design: i:/ME_TOP_LIB/me_top
  Library: ME_TOP_LIB
  Source: /remote/client/db_files/me_top_lib.db


#-----------------------------------------------------------
#       Container : r
#-----------------------------------------------------------

Design: r:/ME_TOP_LIB/me_top
  Library: ME_TOP_LIB
  Source: /remote/client/db_files/me_top_lib.db


1

fm_shell>
```

This example generates a report about all the cells referenced in Implementation container.

```
fm_shell> report_source_path -i
**************************************************
Report         : source_path
```

```
                  -i

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : M-2016.12-SP2-VAL-170316
Date           : Thu Mar 16 03:56:32 2017
**************************************************
#-----------------------------------------------------------
#       Container : i
#-----------------------------------------------------------

Design: i:/ME_TOP_LIB/me_top
  Library: ME_TOP_LIB
  Source: /remote/client/db_files/me_top_lib.db


Design: i:/WORK/top
  Library: WORK
  Source: /remote/client/rtl_out/design.out.v


1
fm_shell>
```

This example creates a report about designs or instances matched by ObjectID list.

```
fm_shell>report_source_path r:/WORK/top/forloop_me\[0*
**************************************************
Report         : source_path
                   r:/WORK/top/forloop_me[0*

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : M-2016.12-SP3
Date           : Thu Mar 16 03:56:32 2017
**************************************************

r:/WORK/top/forloop_me[0].me_inst
Design: r:/ME_TOP_LIB/me_top
  Library: ME_TOP_LIB
  Source: /remote/client/db_files/me_top_lib.db

1
fm_shell>
```

This example creates a report about designs or instances matched by ObjectID list.

```
fm_shell>current_design r:/WORK/top
fm_shell>report_source_path [get_cells]
**************************************************
Report         : source_path
                    _sel1

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : M-2016.12-SP3
Date           : Thu Mar 16 03:56:32 2017
**************************************************

r:/WORK/top/forloop_me[0].me_inst
Design: r:/ME_TOP_LIB/me_top
  Library: ME_TOP_LIB
  Source: /remote/client/db_files/me_top_lib.db
```

```
    r:/WORK/top/forloop_me[1].me_inst
    Design: r:/ME_TOP_LIB/me_top
      Library: ME_TOP_LIB
      Source: /remote/client/db_files/me_top_lib.db

    1
    fm_shell>
```

SEE ALSO

```
get_cells(2)
get_designs(2)
```

report_status

Reports information about the current status of the verification. This command reports the status of the library cells when used in the library verification mode.

SYNTAX

```
status report_status
   [-pass]
   [-fail]
   [-abort]
   [-last]
   [-short]
```

ARGUMENTS

-pass

Reports information about the library cells that passed the verification. This option is available only in the library verification mode.

-fail

Reports information about the library cells that failed the verification. This option is available only in the library verification mode.

-abort

Reports information about the library cells for which the verification was aborted. This option is available only in the library verification mode.

-last

Reports the status information about the compare points verified during the most recent verification.

-short

Use a short report format.

DESCRIPTION

This command reports the status of the verification. The report includes information on the current reference and implementation designs, the result of the recent verification including the number of failing compare points, the status of the recent diagnosis including the number of error candidates, and user-specified setup information.

Use the -short option to get a report that shows the totals of the number of compare points in the major categories. This option gives a report that is compatible with the report_status command in versions earlier than H-2013.03.

Information about the recent verification and diagnosis is cleared when

- The current reference or implementation designs are changed by the **set_reference** or **set_implementation** commands.
- The reference or implementation design container is removed.

In the library verification mode, if no options are specified, the command reports the status of library verification. The report includes the number of cells that are verified, passed, failed and cells that are aborted during verification.

EXAMPLES

This example shows a verification status report.

```
fm_shell> report_status
**************************************************
Report         : status

Reference      : RTL:/WORK/top
Implementation : NET:/WORK/top
Version        : H-2013.03
Date           : Tue Feb 19 16:28:12 2013
**************************************************

********************************* Verification Results *********************************
Verification SUCCEEDED
----------------------
 Reference design: RTL:/WORK/top
 Implementation design: NET:/WORK/top
 5578 Passing compare points
----------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
----------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0     834    4744       0    5578
Failing (not equivalent)       0       0       0       0       0       0       0       0
Not Compared
  Unread                       0       0       0       0       0       4       0       4
****************************************************************************************
1
```

This is the -short version of the same report

```
fm_shell (verify)> report_status -short
**************************************************
Report         : status
                 -short

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : H-2013.03
Date           : Tue Feb 19 16:28:12 2013
**************************************************

Reference Design:      RTL:/WORK/top
Implementation Design: NET:/WORK/top
Verification
   Status:             SUCCEEDED
   Passing Points:     5578
   Failing Points:     0
   Aborted Points:     0
   Unverified Points:  0
Diagnosis
   Status:             Not Run
   Error Candidates:   0
User Defined Setup
   Equivalences:       0
   Compare Points:     0
   Constants:          0
1
```

This example shows a library verification status status report when the **report_status** command is used in library verification mode.

```
fm_shell (library_verify)> report_status
**************************************************
Report         : status

Reference      : r:/WORK/reference
Implementation : i:/WORK/implementation
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

************************* Library Verification Results *************************
Total Cells verified     :    1
Number of Passing Cells  :    0
Number of Failing Cells  :    1
Number of Aborting Cells :    0

fm_shell (library_verify)>
fm_shell (library_verify)> report_status -f
**************************************************
Report         : status

Reference      : r:/WORK/reference
Implementation : i:/WORK/implementation
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

################################################################################
####    Failing Cells
################################################################################
bad_cell
```

```
      Total cells found: 1
```

SEE ALSO

```
diagnose(2)
set_compare_point(2)
set_constant(2)
set_equivalence(2)
set_reference_design(2)
set_implementation_design(2)
verify(2)
```

report_svf_operation

Reports information about automated setup (SVF) operations. You can also report operations in specified compare point's fan-in.

SYNTAX

```
report_svf_operation
   [ -command command_name]
   [ -except_command command_namfP ]
   [ -status status_name ]
   [ -guide ]
   [ -message ]
   [ -summary ]
   id_list | compare_points_list
```

ARGUMENTS

-command *command_name*

Specifies the guide command to search for. Do not specify the "guide_" prefix of the command name. The option may be repeated to search for multiple commands. The guide_transformation sub-types may be used: map, merge, share, tree.

-except_command *command_name*

Specifies the guide command to skip during searches. Do not specify the "guide_" prefix of the command name. The option may be repeated to search for multiple commands. The guide_transformation sub-types may be used: map, merge, share, tree.

-status *status_name*

Specifies the status to search for. The list returned includes only ID numbers for commands that have the specified status. The option may be repeated to search for multiple statuses. Statuses are:

* unprocessed - The operation has been read but not yet processed.

* accepted - The operation has been successfully processed and applied.

* rejected - The processing of the operation failed and the operation was not applied.

* unsupported - The operation is no longer or not yet supported by this release of Formality.

* unaccepted - Any status other than accepted.

-guide

Reports only the text of the commands.

-message

Reports only the warning messages associated with the operations.

-summary

Reports a tabular summary of operations with each id number, line number, command name, and status.

id_list* | *compare_points_list

Specifies a list of ID numbers of the operations to be reported or list of compare points whose fan-in will be searched to find SVF operations. Use **find_svf_operation**, q.v., to generate a list of id numbers of selected operations.

DESCRIPTION

Use this command to print information about SVF operations.

You can optionally specify a list of compare points and report SVF operations in their fan-in cone.

EXAMPLES

```
fm_shell (verify)> report_svf_operation -summary -status rejected
Operation     Line  Command              Status
-----------------------------------------------------
        3       22  transformation_tree   rejected
        9       95  change_names          rejected
       10      114  change_names          rejected
       11      133  change_names          rejected
       12      153  change_names          rejected
1

fm_shell (verify)> report_svf_operation -summary -status rejected
     -except_command change_names
Operation     Line  Command              Status
-----------------------------------------------------
        3       22  transformation_tree   rejected
1

fm_shell (verify)> report_svf_operation -status rejected -command tree

SVF Operation 3 (Line: 22) - transformation_tree.  Status: rejected
## Operation Id: 3
```

```
guide_transformation \
  -design { test } \
  -type { tree } \
  -input { 16 src1 } \
  -input { 16 src2 } \
  -input { 16 src4 } \
  -input { 16 src6 } \
  -input { 16 src8 } \
  -output { 18 O1 } \
  -pre_resource { { 17 } add_5 = UADD { { src1 ZERO 17 } { src2 ZERO 17 } } } \
  -pre_resource { { 18 } add_5_2 = UADD { { add_5 ZERO 18 } { src4 ZERO 18 } } } \
  -pre_resource { { 18 } add_5_3 = UADD { { add_5_2 } { src6 ZERO 18 } } } \
  -pre_resource { { 18 } sub_5 = USUB { { add_5_3 } { src8 ZERO 18 } } } \
  -pre_assign { O1 = { sub_5 } } \
  -post_resource { { 18 } add_3_root_sub_5 = UADD { { src1 ZERO 18 }
     { src2 ZERO 18 } } } \
  -post_resource { { 18 } add_2_root_sub_5 = UADD { { src4 ZERO 18 }
     { src6 ZERO 18 } } } \
  -post_resource { { 18 } add_1_root_sub_5 = UADD { { add_3_root_sub_5 }
     { add_2_root_sub_5 } } } \
  -post_resource { { 18 } sub_0_root_sub_5 = USUB { { add_1_root_sub_5 }
     { src8 ZERO 18 } } } \
  -post_assign { O1 = { sub_0_root_sub_5 } }

Info:  guide_transformation 3 (Line: 22)  Could not find pre_resource 'sub_5' in design 'test'.

1

fm_shell (verify)> report_svf_operation -message -status rejected -command tree

Info:  guide_transformation 3 (Line: 22)  Could not find pre_resource 'sub_5' in design 'test'.

1

fm_shell (verify)> report_svf_operation -guide -status rejected -command tree

SVF Operation 3 (Line: 22) - transformation_tree.  Status: rejected
## Operation Id: 3
guide_transformation \
  -design { test } \
  -type { tree } \
  -input { 16 src1 } \
  -input { 16 src2 } \
  -input { 16 src4 } \
  -input { 16 src6 } \
  -input { 16 src8 } \
  -output { 18 O1 } \
  -pre_resource { { 17 } add_5 = UADD { { src1 ZERO 17 } { src2 ZERO 17 } } } \
  -pre_resource { { 18 } add_5_2 = UADD { { add_5 ZERO 18 } { src4 ZERO 18 } } } \
  -pre_resource { { 18 } add_5_3 = UADD { { add_5_2 } { src6 ZERO 18 } } } \
  -pre_resource { { 18 } sub_5 = USUB { { add_5_3 } { src8 ZERO 18 } } } \
  -pre_assign { O1 = { sub_5 } } \
  -post_resource { { 18 } add_3_root_sub_5 = UADD { { src1 ZERO 18 }
     { src2 ZERO 18 } } } \
  -post_resource { { 18 } add_2_root_sub_5 = UADD { { src4 ZERO 18 }
     { src6 ZERO 18 } } } \
  -post_resource { { 18 } add_1_root_sub_5 = UADD { { add_3_root_sub_5 }
     { add_2_root_sub_5 } } } \
  -post_resource { { 18 } sub_0_root_sub_5 = USUB { { add_1_root_sub_5 }
     { src8 ZERO 18 } } } \
  -post_assign { O1 = { sub_0_root_sub_5 } }

1
```

```
fm_shell (verify)> report_svf_operation {r:/WORK/dp/O3[1] r:/WORK/dp/O3[4]}
     -command merge

## SVF Operation 5 (Line: 47) - transformation_merge.  Status: rejected
## Operation Id: 5
guide_transformation \
  -design { dp } \
  -type { merge } \
  -input { 4 I1 } \
  -input { 4 I2 } \
  -input { 4 I3 } \
  -input { 4 I4 } \
  -output { 8 O1 } \
  -pre_resource { { 8 } mult_12 = MULT_TC { { I1 } { I2 } { 0 } } } \
  -pre_resource { { 8 } mult_12_2 = MULT_TC { { I3 } { I4 } { 0 } } } \
  -pre_resource { { 8 } add_12 = ADD { { mult_12 } { mult_12_2 } } } \
  -pre_assign { O1 = { add_12 } } \
  -datapath add_12_DP_OP_258_7518_1

Info:  guide_transformation 5 (Line: 47) Retrying with unread processing enabled.
Info:  guide_transformation 5 (Line: 47)  Could not find pre_resource 'mult_12' in design 'dp'.
Info:  guide_transformation 5 (Line: 47)  Could not find pre_resource 'mult_12_2' in design 'dp'.
Info:  guide_transformation 5 (Line: 47)  Could not find pre_resource 'add_12' in design 'dp'.

1


fm_shell (verify)> report_svf_operation r:/WORK/dp/O*  -status rejected -command datapath

## SVF Operation 10 (Line: 116) - datapath.  Status: rejected
## Operation Id: 10
guide_datapath \
  -design { dp } \
  -datapath { add_12_DP_OP_258_7518_1 } \
  -body { dp_add_12_DP_OP_258_7518_0 }

Info:  guide_datapath 10 (Line: 116)  Pre-verification of
r:/WORK/dp/add_12_DP_OP_258_7518_1 INCONCLUSIVE.

1
```

SEE ALSO

```
find_svf_operation(2)
set_svf(2)
report_guidance(2)
remove_guidance(2)
```

report_truth_table

Generate and print truth table for given signal.

SYNTAX

```
report_truth_table [ signalId ]
   [ -display_fanin ]
   [ -fanin { list of signals }  ]
   [ -constraint {signal=[0/1]}+  ]
   [ -nb_lines int ]
   [ -max_line int ]
   [ -max_fanin int ]
```

ARGUMENTS

signalId

Name of signal(port/net/pin) for which to generate the truth table. Name must be valid Formality signal Id.

-display_fanin

This option is used to view only the fanin for given signal. No truth table is generated.

-fanin {signal signal ...}

This option is used to view the truth table for a given signal in terms of the specified fanin signals. This option can be used to change the default order of the fanin printed in the truth table. The other use of this option is to limit the truth table to only those fanins that may be of interest to the user. In this case,the output will be expressed as a logical equation in terms of the unspecified fanin signals.

-constraint {signal=0/1 signal=0/1 ...}

This option allows the user to constraint fanin signals to either 0 or 1. This will help in reducing the size of the truth table.

-nb_lines integer

This option sets maximum size of truth table row. This is used to limit the size of logical expression for specified signal. Default value for this size is 4096.

-max_line integer

This option sets maximum number of lines that can be printed for the truth table. Default value is 512.

-max_line integer

This option sets maximum fanin size that can be printed for the truth table. Default value is 25.

DESCRIPTION

This command is used to view the truth table for given signal. It enhances the debug environment in Formality and can be used specially for smaller cones or library cells. This feature provides user the ability to vire truth table for failing points in terms of desired signal fanin. There is also an option to analyze the output value by constraining fanin signals to 1 or 0.

The **report_truth_table** command returns the following:

```
  * 0 for failure
  * 1 for success
```

EXAMPLES

* View truth table for a two-input and cell

```
(fm_shell) report_truth_table r:/WORK/my_and/out
 Truth table for signal : "out"
 ----------------------
 in1
 | in2
 | |
 0 . | 0
 1 0 | 0
 1 1 | 1

 ******* End truth table *******
```

* View truth table for D-flip flop

```
(fm_shell) report_truth_table r:/WORK/dff/q
 Truth table for signal : "q"
 ----------------------
 clk<1>
 | clk
 | | d
 | | |
 0 0 . | q<1>
 0 1 0 | 0
 0 1 1 | 1
 1 . . | q<1>

 ******* End truth table *******
```

* View truth table for 4-input and gate in terms of chosen fanin

```
(fm_shell) report_truth_table -fanin { a b } r:/WORK/my_and/out
 Truth table for signal : "out"
 ----------------------
 a
 | b
 | |
 0 . | 0
 1 0 | 0
 1 1 | c & d

 ******* End truth table *******
```

* View truth table for 2-input xor by constraining one signal

```
(fm_shell) report_truth_table -constraint {y=1} r:/WORK/my_xor/out
 Truth table for signal : "out"
 ----------------------
 y = 1
 z
 |
 0 | 1
 1 | 0

 ******* End truth table *******
```

SEE ALSO

report_undriven_nets

Reports the nets that are undriven after match.

SYNTAX

```
report_undriven_nets
   [-substring substring]
   [-reference]
   [-implementation]
```

Data Types

```
substring string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-substring *substring*

Reports only the nets containing the specified substring.

-reference

Reports the undriven nets in the reference design.

-implementation

Reports the undriven nets in the implementation design.

DESCRIPTION

This command reports the nets that are undriven after the most recent match. The report includes the total number of undriven nets, followed by a list of undriven nets.

EXAMPLES

The following example is a report generated by the **report_undriven_nets** command.

```
prompt> report_undriven_nets
**************************************************
Report         : undriven_nets

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

4 Undriven nets:

r:/WORK/top/out_or
r:/WORK/top/or_one.a
i:/WORK/top/and_zero.b
i:/WORK/top/out_or
1
```

SEE ALSO

```
report_multidriven_nets(2)
```

report_unmatched_points

Reports the unmatched design objects after using the **match** command.

SYNTAX

```
int report_unmatched_points
   [-compare_rule]
   [-substring substring]
   [-point_type point_type]
   [-inputs input_type]
   [-status status]
   [-except_status except_status]
   [-reference]
   [-implementation]
   [-datapath]
   [-list]
   [-never_loads]
   [-always_reset]
   [object_list [-type type]]
```

Data Types

```
substring string
point_type string
input_type string
status string
except_status string
object_list string
type string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-compare_rule

Reports mapped object names using the compare rules.

-substring *substring*

Reports compare points with the specified substring.

-point_type *point_type*

Reports the specified types of compare points. By default, input points and compare points of all types are included in the report. You need to specify one of the following point types:

- *all* - Reports all object types.
- *bbox* - Reports black boxes.
- *bbox_pin* - Reports all black-box pins.
- *bbox_input* - Reports black-box input pins.
- *bbox_output* - Reports black box output pins.
- *bbox_inout* - Reports black box bidirectional pins.
- *block* - Reports hierarchical blocks.
- *block_pin* - Reports hierarchical block pins.
- *block_input* - Reports hierarchical block input pins.
- *block_output* - Reports hierarchical block output pins.
- *block_inout* - Reports hierarchical block bidirectional pins.
- *loop* - Reports loop points.
- *net* - Reports net points.
- *cut* - Reports cut points. Includes Und and Unk cone inputs.
- *bbox_net* - Reports resolved black box and multiply-driven nets.
- *port* - Reports all ports.
- *input* - Reports input ports.
- *output* - Reports output ports.
- *inout* - Reports bidirectional ports.
- *directly_undriven_output* - Reports output and bidirectional ports without connected nets or with connected nets but without a driving pin.
- *reg* - Reports all registers.
- *DFF* - Reports flip-flop registers.
- *LAT* - Reports latch registers.
- *PDCut* - Reports power domain pin compare points.
- *PGPin* - Reports power, ground, or power-down function compare points.

-inputs *input_type*

Reports points with undriven or unmatched inputs.

-status *status*

Reports the points that match the specified status. If both the **-status** and **-point_type** options are specified, the reported points must pass both filters.

By default, the **report_unmatched_points** command does not report the points with the *unread* status. The command also does not report the unmatched *dont_verify* points resulting in verification failure. To suppress the reporting of all *dont_verify* points, use the **-except_status dont_verify** option.

The **-status** option can have one of the following arguments:

- *none* - Reports all except these special types.
- *trans* - Reports transparent latch registers.
- *const* - Reports all constant registers.
- *const0* - Reports constant 0 registers.
- *const1* - Reports constant 1 registers.
- *constX* - Reports constant X registers.
- *const0X* - Reports constrained 0X registers.
- *const1X* - Reports constrained 1X registers.
- *clock_gate* - Reports clock-gate latches.
- *dont_verify* - Reports dont_verify points.
- *unread* - Reports unread points.
- *undriven* - Reports undriven points.
- *targeted* - Reports points targeted for verification (requires the "Formality-Ultra" license key).
- *not_targeted* - Reports points that are not targeted for verification (requires the "Formality-Ultra" license key).

-except_status *except_status*

Excludes points that match the specified type from the report.

You need to specify one of the following values for the *except_status* argument:

- *trans* - Excludes transparent latch registers.
- *const* - Excludes all constant registers.
- *const0* - Excludes constant 0 registers.
- *const1* - Excludes constant 1 registers.

- *constX* - Excludes constant X registers.
- *const0X* - Excludes constrained 0X registers.
- *const1X* - Excludes constrained 1X registers.
- *clock_gate* - Excludes clock-gate latches.
- *dont_verify* - Excludes all *dont_verify* points. By default, the report includes unmatched dont_verify points that might result in verification failure.
- *unread* - Excludes unread points.
- *undriven* - Excludes undriven points.
- *targeted* - Excludes points targeted for verification (requires the "Formality-Ultra" license key).
- *not_targeted* - Excludes points not targeted for verification (requires the "Formality-Ultra" license key).

`-reference`

Reports compare points in the reference design. By default, the **report_unmatched_points** command reports compare points in both the reference and the implementation designs.

`-implementation`

Reports compare points in the implementation design. By default, the **report_unmatched_points** command reports compare points in both the reference and the implementation designs.

`-datapath`

Reports datapath blocks in the reference design.

`-list`

Returns a list of point names which can be used in further Tcl processing.

`-never_loads`

Filters the report to only include register compare points that do not synchronously load.

`-always_reset`

Filters the report to only include register compare points that are constantly reset.

`object_list`* `-type` *`type`

Reports unmatched cone inputs in the fanin of the compare point specified by the *object_list* option. Use the **-type** *ID_type* option to specify the type of the compare point if its name is ambiguous. The **-type** *ID_type* option is valid only when you use the *objectID* argument.

DESCRIPTION

This command reports the design objects that are not matched. The report includes the total number of unmatched points followed by a list of unmatched design objects.

EXAMPLES

The following command reports the unmatched design objects in both the reference and the implementation designs.

```
fm_shell> report_unmatched_points
**************************************************
Report         : unmatched_points

Reference      : HDL:/WORK/CORE
Implementation : SCAN:/WORK/CORE
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

3 Unmatched points (0 reference, 3 implementation):

  Impl Port SCAN:/WORK/CORE/test_se

  Impl Port SCAN:/WORK/CORE/test_si

  Impl Port SCAN:/WORK/CORE/test_so

 [BBNet: multiply-driven net
  BBox:  black box
  BBPin: black box pin
  Block: hierarchical block
  BlPin: hierarchical block pin
  Cut:   cut-point
  DFF:   non-constant DFF register
  DFF0:  constant 0 DFF register
  TLA:   transparent latch register
  TLA0X: transparent constrained 0X latch register
  TLA1X: transparent constrained 1X latch register
  Loop:  cycle break point
  Net:   matchable net
  Port:  primary (top-level) port
  Und:   undriven signal cut-point]
1
```

SEE ALSO

```
report_aborted_points(2)
report_failing_points(2)
report_matched_points(2)
report_passing_points(2)
report_unverified_points(2)
```

report_unread_endpoints

Reports the end objects that cause points to become unread.

SYNTAX

```
int report_unread_endpoints
   [-all]
   [-point_type point_type]
   [-cause cause]
   [unread_points_list]
```

Data Types

```
point_type string
cause string
unread_points_list string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-all

Reports all unread points in the design.

-point_type *point_type*

Reports the unread points of the specified point type argument. By default, unread input points and compare points of all types except block and block pin are included.

Specify one of the following arguments for the **point_type** argument:

- *bbox_pin* - Reports unread black box pins.
- *bbox_input* - Reports unread black box input pins.

- *bbox_output* - Reports unread black box output pins.
- *bbox_inout* - Reports unread black box bidirectional pins.
- *loop* - Reports unread loop points.
- *cut* - Reports unread cutpoints.
- *bbox_net* - Reports unread black-box resolved and multiply-driven nets.
- *port* - Reports all unread ports.
- *input* - Reports unread input ports.
- *inout* - Reports unread bidirectional ports.
- *reg* - Reports all unread registers.
- *DFF* - Reports unread flip-flop registers.
- *LAT* - Reports unread latch registers.

-cause *cause*

Reports the end points that match the specified cause argument.

Specify one of the following **cause** arguments for the option:

- *no_reader* - Reports end points due to no reader.
- *constant* - Reports end points due to blocking constant.

unread_points_list

Reports the specified unread points. The format of the list is `{cp1 cp2}`. Use wildcard characters to match multiple points.

DESCRIPTION

This command reports unread end points. It reports end objects which cause points to become unread. The report lists each end point, its type, and the cause for being a path terminating object.

EXAMPLES

The following example reports a specific unread register.

```
fm_shell> report_unread_endpoints r:/WORK/top/q3_reg
**********************************************
Report         : unread_endpoints
```

```
Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

Following 2 blocking objects identified in the fanout:

    (Net )  r:/WORK/top/fm_m21/andOut0[1]  (blocked by constant)
    (Net )  r:/WORK/top/q2n  (no reader)
```

The following example reports all unread registers in the design.

```
fm_shell> report_unread_endpoints  -all -point_type reg
**************************************************
Report         : unread_endpoints

Reference      : r:/WORK/top
Implementation : i:/WORK/top
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

Following 5 blocking objects identified in the fanout:

    (Cell)  r:/WORK/top/q5_reg  (no reader)
    (Net )  r:/WORK/top/fm_m13/andOut0[0]  (blocked by constant)
    (Net )  r:/WORK/top/fm_m21/andOut0[1]  (blocked by constant)
    (Net )  r:/WORK/top/q2n  (no reader)
    (Net )  i:/WORK/top/fm_m10/andOut0[0]  (blocked by constant)
```

SEE ALSO

```
report_matched_points(2)
report_unmatched_points(2)
diagnose(2)
```

report_unverified_points

Reports the compare points that must be verified again.

SYNTAX

```
status report_unverified_points
   [-compare_rule]
   [-substring string]
   [-point_type point_type]
   [-inputs input_type]
   [-status status]
   [-cause cause]
   [-list]
   [-last]
   [-never_loads]
   [-always_reset]
```

Data Types

```
string string
point_type string
input_type string
status string
cause string
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-compare_rule

Reports objects that are mapped using compare rules.

-substring *string*

Reports compare points containing the specified string argument.

-point_type *point_type*

Reports the compare points that are of the specified point type argument. By default, all compare point types are included.

Specify one or more of the following point types:

- *bbox_pin* - Reports all black-box pin compare points.
- *bbox_input* - Reports black-box input pin compare points.
- *bbox_inout* - Reports black-box bidirectional pin compare points.
- *loop* - Reports loop compare points.
- *bbox_net* - Reports black-box resolved and multiply-driven nets.
- *cut* - Reports cutpoints.
- *port* - Reports all port compare points.
- *output* - Reports output port compare points.
- *inout* - Reports bidirectional port compare points.
- *directly_undriven_output* - Reports output and bidirectional ports that do not have a connected net, or that have a connected net but do not have a driving pin.
- *reg* - Reports all register compare points.
- *DFF* - Reports flip-flop register compare points.
- *LAT* - Reports latch register compare points.
- *trans* - Reports transparent latch register compare points.
- *PDCut* - Reports power domain pin compare points.
- *PGPin* - Reports power, ground, or power-down function compare points.

-inputs *input_type*

Reports compare points with input points that are undriven or unmatched.

Specify one of the following input type arguments for the option:

- *undriven* - Reports compare points with undriven input points
- *unmatched* - Reports compare points with unmatched input points

-status *status*

Reports compare points with the specified polarity arguments. By default, compare points with either polarities are reported.

Specify one of the following polarities:

- *inverted* - Reports compare points with inverted match polarity
- *noninverted* - Reports compare points with noninverted match polarity

-cause ***cause***

Reports the compare points whose unverified status is due to the specified cause. Specify one of the following values for the *cause* argument:

- *interrupt* - Reports compare points that are unverified due to an interrupt (^C) or timeout.
- *fail_limit* - Reports compare points that are unverified because failing point limit was reached.
- *matching* - Reports compare points that are unverified because of matching changes.

-list

Reports a list of pairs of point names that can be used in further Tcl processing.

-last

Reports the points that are unverified in the previous verification.

-never_loads

Filters the report to only include register compare point pairs that do not synchronously load.

-always_reset

Filters the report to only include register compare point pairs that are constantly reset.

DESCRIPTION

This command reports compare points that are not verified.

The reports includes the total number of unverified compare points in each of the categories, followed by the design objects that represent the unverified compare point.

EXAMPLES

```
fm_shell> report_unverified_points
**************************************************
Report         : unverified_points

Reference      : r:/WORK/reference
Implementation : i:/WORK/implementation
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

7 Unverified compare points:
       0 unverified because of interrupt or timeout
       7 unverified because failing point limit reached
       0 affected by matching changes
```

```
      Ref DFF        r:/WORK/reference/ffA/q_reg[0]
      Impl DFF        i:/WORK/implementation/ffA/q_reg[0]

      Ref DFF        r:/WORK/reference/ffB/q_reg[0]
      Impl DFF        i:/WORK/implementation/ffB/q_reg[0]

      Ref DFF        r:/WORK/reference/ffB/q_reg[1]
      Impl DFF        i:/WORK/implementation/ffB/q_reg[1]

      Ref DFF        r:/WORK/reference/ffC/q_reg[0]
      Impl DFF        i:/WORK/implementation/ffC/q_reg[0]

      Ref DFF        r:/WORK/reference/ffC/q_reg[1]
      Impl DFF        i:/WORK/implementation/ffC/q_reg[1]

      Ref DFF        r:/WORK/reference/ffC/q_reg[2]
      Impl DFF        i:/WORK/implementation/ffC/q_reg[2]

      Ref DFF        r:/WORK/reference/ffC/q_reg[3]
      Impl DFF        i:/WORK/implementation/ffC/q_reg[3]

     [BBNet: multiply-driven net
      BBPin: black box pin
      Cut:   cut-point
      DFF:   non-constant DFF register
      DFF0:  constant 0 DFF register
      DFF1:  constant 1 DFF register
      DFFX:  constant X DFF register
      DFF0X: constrained 0X DFF register
      DFF1X: constrained 1X DFF register
      TLA:   transparent latch register
      TLA0X: transparent constrained 0X latch register
      TLA1X: transparent constrained 1X latch register
      Loop:  cycle break point
      Port:  primary (top-level) port
      Und:   undriven signal cut-point]
    1
```

SEE ALSO

```
report_failing_points(2)
report_passing_points(2)
report_aborted_points(2)
report_matched_points(2)
report_unmatched_points(2)
report_not_compared_points(2)
```

report_upf

Reports information on the UPF objects that are inserted into the designs using the **load_upf** command.

SYNTAX

```
status report_upf
   [-r]
   [-i]
   [-container container_name]
   [-verbose]
   [-isolation]
   [-retention]
   [-power_switch]
   [-pst]
   [-instance_paths]
```

Data Types

```
      container_name string
```

Enabled Shell Modes

Setup
Match
Verify

ARGUMENTS

-r

Reports the UPF information of the reference container. By default, the **report_upf** command reports information on all containers.

-i

Reports UPF information of the implementation container. By default, the **report_upf** command reports information on all containers.

-container *container_name*

Specifies a container to report.

-verbose

Reports detailed information about the loaded files, the number of cells inserted, and the implemented power-state tables. Specifying the following options at the same time is the same as using the *-verbose* option: **-retention**, **-isolation**, **-power_switch**, **-pst**.

-isolation

Reports all the isolation cells that are inserted by each power domain and the isolation strategy they came from.

-retention

Reports all the retention cells that are inserted by Formality by each power domain and the retention strategy they came from.

-power_switch

Reports the power switch cells that are inserted into the design along with their power domain name and output supply net.

-pst

Reports summary information for loaded power-state tables. This option reports the total number of supplies and states for each table.

-instance_paths

Reports the instance path names for cells in addition to the folded design path names which are the default. This option will affect the **-verbose**, **-isolation**, **-retention**, and **-power_switch** options. The **-instance_paths** option is only available after match.

DESCRIPTION

This command reports a summary of UPF information of all containers. When data is reported for a container, it begins with the list of UPF files that were loaded followed by the setting of the **upf_implementation_based_on_file_headers** variable and file comments. Totals are reported for the number of inserted isolation cells, retention cells, power switch cells, and power-state tables created.

When **-isolation**, **-retention**, **-power_switch**, or **-pst** options are specified, the inferred cells or UPF objects are grouped and listed after each UPF file they originated from. The UPF file line number of the creation command is reported for each object. Isolation and retention cells have a heading with the total number of cells inserted for the specified upf file. Then it has a sub-heading for each strategy, which displays the UPF file line number, power domain name, strategy name, and total cells inserted for this strategy. That will be followed by the folded design path name to each inferred cell for the given strategy. UPF power switch cells have a heading with the total number of switch cells inserted for the specified UPF file. Then a sub-heading containing the UPF file line number, domain name, and switch name is displayed.

This is followed by the folded design path to the UPF switch cell, a sub-heading: Output supply net, and then the path name to the switch output supply net.

The **-instance_paths** option is available after matching. When this option is specified, the command reports the instance pathnames of all three cell types in addition to the folded design path names. Power state tables are displayed with a heading indicating the total number of tables created for the UPF file. A sub-heading for each power state table displays the UPF file line number of the creation command, and power state table name. This will be followed by the total number of supplies and states for each power state table.

EXAMPLES

The following example reports UPF summary information on the default reference and implementation containers.

```
Container:   r

UPF file loaded and file control variable value:
------------------------------------------------
/slowfs/dept5138v/sscherr/work/upf/report/domain_merge/top_u5.upf
        upf_implementation_based_on_file_headers: true

Power State Tables created: 0

Isolation cells inserted: 0

Retention cells inserted: 0

UPF power switch cells inserted: 0


UPF file loaded and file control variable value:
------------------------------------------------
/slowfs/dept5138v/sscherr/work/upf/report/domain_merge/u51.upf
        upf_implementation_based_on_file_headers: true

Power State Tables created: 1

Isolation cells inserted: 0

Retention cells inserted: 1

UPF power switch cells inserted: 1


UPF file loaded and file control variable value:
------------------------------------------------
/slowfs/dept5138v/sscherr/work/upf/report/domain_merge/top_only1.upf
        upf_implementation_based_on_file_headers: true

Power State Tables created: 2

Isolation cells inserted: 24

Retention cells inserted: 7
```

```
      UPF power switch cells inserted: 5


      load_upf has not been run on container: i.
      1
```

SEE ALSO

```
load_upf(2)
upf_implementation_based_on_file_headers(3)
```

report_user_matches

Reports the user-defined matched points.

SYNTAX

```
status report_user_matches
   [-inverted | -noninverted | -unknown]
```

ARGUMENTS

-inverted

Reports the user-defined matches that have an inverted polarity. This option is mutually exclusive to the *-noninverted* and *-unknown* options.

-noninverted

Reports the user-defined matches that have the regular polarity. This option is mutually exclusive to the *-inverted* and *-unknown* options.

-unknown

Reports the user-defined matches that have an unknown polarity. This option is mutually exclusive to the *-inverted* and *-noninverted* options.

DESCRIPTION

This command reports compare points that match when the **set_user_match** command is specified and returns a list of user-defined matched compare point pairs.

User-defined matches are not applied to the designs until the *match* or *verify* commands are run. Other related points are matched as a result of a user-defined match. After a user-defined match is applied, you can report on the affected points with the *report_matched_points -method user* command.

EXAMPLES

The following example reports the user-defined matched points.

```
fm_shell (setup)> report_user_matches
**************************************************
Report         : user_matches

Reference      : spec:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

User matched points:

    (Port)       impl:/WORK/UPC/UPC_DATA[4]
    (Port)       spec:/WORK/UPC/UPC_DATA[5]

    (Port)       spec:/WORK/UPC/UPC_DATA[4]
    (Port)   (-) impl:/WORK/UPC/UPC_DATA[5]

    (BBPin)      impl:/WORK/UPC/INT_UPC_reg[3]
    (BBPin)  (?) spec:/WORK/UPC/INT_UPC_reg[3]
```

SEE ALSO

```
remove_compare_point(2)
report_matched_points(2)
set_compare_point(2)
set_user_match(2)
```

report_verify_points

Reports information about the user-defined verify points.

SYNTAX

```
status report_verify_points
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command reports the points that are defined as verify points using the *set_verify_points* command.

The command also lists the current verification status, if known, of each point.

EXAMPLES

The following example shows how to use the command. Each line represents a design object.

```
fm_shell (verify)> report_verify_points
Verify points:
    (PASSED)    (Port)    r:/WORK/top/o3
    (PASSED)    (Port)    r:/WORK/top/o4
```

SEE ALSO

```
remove_verify_points(2)
set_verify_points(2)
```

report_vhdl

Reports a list of VHDL configurations, entities, architectures, and associated generics.

SYNTAX

```
status report_vhdl
   [-configuration]
   [-entity]
   [-package]
```

ARGUMENTS

-configuration

Reports the top-level entities and the associated architectures.

-entity

Reports the entities, including their architectures and generics with their default values.

-package

Reports the packages that are loaded in the current design.

DESCRIPTION

This command reports a list of VHDL configurations, entities, generics and architectures in the Formality environment.

EXAMPLES

The following example reports all VHDL objects in the Formality environment.

```
fm_shell> report_vhdl
**************************************************
Report         : vhdl

Reference      : ref:/WORK
Implementation : impl:/WORK
Version        : G-2012.06
Date           : Wed Jun 6 12:46:37 2012
**************************************************

   **********  Container ref  **********
   loaded entities:
   ----------------
   ref:/WORK/FULL_ADDER
   Generics:
   None
   Architectures:
   NEW_FULL_ADDER
   REGULAR_FULL_ADDER
   ref:/WORK/HALF_ADDER
   Generics:
   None
   Architectures:
   REGULAR_HALF_ADDER
   NEW_HALF_ADDER
   ref:/WORK/generic2
   Generics:
   n = No Default
   Architectures:
   GENERIC2
   loaded configurations:
   ----------------------
   ref:/WORK/ConfigAdd
    FULL_ADDER
    Regular_Full_Adder
   ref:/WORK/ConfigHalf
   HALF_ADDER
   New_Half_Adder
   loaded packages:
   ----------------------
   ref:/WORK/PACK_PACK100
1
```

SEE ALSO

restore_session

Restores a saved Formality session.

SYNTAX

```
status restore_session
   file
```

Data Types

```
   file string
```

ARGUMENTS

file_name

Specifies a file that is previously saved by the **save_session** command or using the Formality GUI.

DESCRIPTION

This command restores a previously saved Formality session. When you restore a session, all information in the current Formality session is lost, unless the current session is already saved.

Restored information includes design and technology libraries, current reference and implementation designs, design and environment parameters, and verification and diagnosis status.

EXAMPLES

The following example restores information from a session file named *synth_test*, in which verification succeeded.

```
fm_shell> restore_session synth_test.fss
Session restored.
Last Verification Status:
reference design : s:/WORK/CORE
implementation design: i:/WORK/CORE
Verification SUCCEEDED
Diagnosis - N/A
```

The following example restores information from a session file named *synth_test_2*, in which the implementation and reference containers are not established.

```
fm_shell> restore_session synth_test_2.fss
Session restored.
Last Verification Status: Verification not run.
```

SEE ALSO

```
save_session(2)
```

rewire_connection

Rewires the input pins driven by one object so that they are driven by another object.

SYNTAX

```
status rewire_connection
   [-type ID_type_list]
   [-invert]
   -from ID1 -to ID2
```

Data Types

```
   ID_type_list string
   ID1 string
   ID2 string
```

ARGUMENTS

-type *ID_type_list*

Specifies the type of the specified objects. Use this option if the design contains more than one type of object with the specified name. Specify one of the following object types:

- *port* - Specifies that both objects are ports
- *pin* - Specifies that both objects are pins
- *net* - Specifies that both objects are nets
- *{pin net}* - Specifies that the first object is a pin and the second is a net
- *{port net}* - Specifies that the first object is a port and the second is a net
- *{net pin}* - Specifies that the first object is a net and the second is a pin
- *{net port}* - Specifies that the first object is a net and the second is a port
- *{port pin}* - Specifies that the first object is a port and the second is a pin
- *{pin port}* - Specifies that the first object is a pin and the second is a port

-invert

Inverts the connections.

-from ID1

Specifies the design object that drives the input pins that will be reconnected to the object specified by the *-to ID2* argument. The object specified by the *-from ID1* argument must be one of the following:

- input or inout port
- output or inout pin
- net

-to ID2

Specifies the new driver of the input pins driven by the object specified by the *-from ID1* argument. The object specified by the *-to ID2* argument must be one of the following:

- input or inout port
- output or inout pin
- net

DESCRIPTION

This command rewires the design that contains the specified objects. The specified objects must belong to the same design and must be one of the following:

- input or inout port
- output or inout pin
- net
  The input pins driven by the object specified with the *-from ID1* argument are disconnected from their net and reconnected to the net driven by the object specified with the *-to ID2* argument.

EXAMPLES

This example shows how to reconnect the input pins driven by port P2 to be driven by port P1 instead. In this case, the *-type* argument differentiates the port from a similarly named net.

```
fm_shell> rewire_connection -type port -from ref:/WORK/CORE/P2 -to ref:/WORK/CORE/P1
Rewired port 'P1' and port 'P2'
1
```

SEE ALSO

run_alternate_strategies

Runs alternate strategies in parallel.

SYNTAX

```
status run_alternate_strategies
   [-directory pathname ]
   [-replace]
   [-session session_file ]
   [-tclfile tcl_file ]
   [-strategies strategies_list ]
   [-run_all]
```

Data Types

```
pathname        string
session_file    string
tcl_file        string
strategies_list string
integer         integer
```

ARGUMENTS

-directory *pathname*

Specifies the directory containing data for all the runs.

-replace

Replaces the existing directory.

-session *session_file*

Specifies the session file containing the designs.

-tclfile *tcl_file*

Specifies the Tcl file with scripts to read the designs.

-strategies *strategies_list*

Specifies the list of strategies to run.

-run_all

Runs all specified strategies to completion even if one strategy is successful.

DESCRIPTION

This command automates the process of running alternate strategies in parallel. If one of the alternate strategy successfully completes verification, all other strategy runs are terminated.

All log files and run information is generated within the directory specified using the **-directory** option. If this option is not specified, a unique directory starting with the prefix formality_alternate_strategy is created.

Either a session file or a Tcl file must be specified. If the Tcl file is specified, it must contain the **verify** command.

You can specify the list of strategies to be run using the **-strategies** option. The list of currently supported strategies is stored in the **verification_alternate_strategy_names** variable. If the **-strategies** option is not specified, all the supported strategies are run, except for the default strategy *none*. This is because the command assumes that you have used the default strategy, and the results were inconclusive.

set_run_alternate_strategies_options must be issued prior to **run_alternate_strategies** command. set_run_alternate_strategies_options command specifies the compute farm configuration necessary to start alternate strategies in parallel.

By default, when one of the alternate strategies successfully completes verification, remaining running strategies are killed. Strategies that are pending in the queue are not started. You can use the **-run_all** option to override this behavior. With this option, you can complete running all strategies. This option can also be used to identify a strategy that takes the least amount of resources to verify the design.

By default, alternate strategy master and workers communicate using IPv4 protocol. IPv6 protocol can be enabled through an environment variable CDPL_IPV6.

To enable IPv6 through this variable: setenv CDPL_IPV6

To disable IPv6 or to use IPv4 protocol: unsetenv CDPL_IPV6

Following protocols are supported for launchig jobs on remote machines:

```
LSF
SGE
PBS
RTDA
NB
CUSTOM (users can specify job submit script)
```

EXAMPLES

The following examples illustrate the command usage for running alternate strategies in parallel in different farm environments.

1. SGE farm - 16 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol SGE
   -submit_command "qsub -V -P bnormal -cwd
   -l arch=glinux,os_bit=64,mem_free=10G -pe mt 4" -num_processes 16
fm_shell> run_alternate_strategies -session post_match.fss
```

2. SGE farm - custom script for 16 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol SGE
   -submit_command custom.scr -num_processes 16
fm_shell> run_alternate_strategies -session post_match.fss

custom.scr
#!/bin/csh -f
qsub -V -P bnormal -cwd -l arch=glinux,os_bit=64,mem_free=10G -pe mt 4 $*
```

3. SGE farm - 16 workers each using 4 cores with -tclfile option

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol SGE
   -submit_command "qsub -V -P bnormal -cwd
   -l arch=glinux,os_bit=64,mem_free=10G -pe mt 4" -num_processes 16
fm_shell> run_alternate_strategies -tclfile fm.tcl

fm.tcl
set_svf default.svf
read_container -r ref.fsc
read_container -i impl.fsc
verify
quit
```

4. LSF farm - 8 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol LSF
   -submit_command "bsub -n 4 -R \"qscm\" -R \"rusage\[mem=32000\]\"
   -R \"span\[hosts=1\]\"" -num_processes 8
fm_shell> run_alternate_strategies -session post_match.fss
```

SEE ALSO

```
set_run_alternate_strategies_options(2)
```

save_session

Saves the current Formality session.

SYNTAX

```
status save_session
   [-replace]
   file
```

Data Types

```
file string
```

ARGUMENTS

-replace

Replaces an existing file with the file being saved.

file

Specifies the name of the file in which the Formality session is saved. You can specify the full path name or only the file name. If you do not specify the path, the tool saves the file in the current working directory.

DESCRIPTION

This command saves the current Formality session to a specified .fss file.

The session information includes all open containers and environment parameters, verification results, and diagnosis status. You must have successfully run the **set_top** command on the containers before the session can be saved.

Note: Backward compatibility is not guaranteed for Formality session files. New versions of the tool might not support session files generated from earlier versions.

EXAMPLES

The following example writes the Formality session information to the file named *regression_out* in the $TEST directory.

```
fm_shell> save_session $TEST/regression_out
Info: wrote file '/d/atg2/test/regression_out.fss'
```

SEE ALSO

```
restore_session(2)
```

select_cell_list

Selects library cells depending on the option specified. This command is available only in the library verification mode.

SYNTAX

```
status select_cell_list
   [-add cell_names ]
   [-remove cell_names ]
   [-clear]
   [-file file_name]
   [cell_names]
```

Data Types

```
cell_names string
file_name string
```

ARGUMENTS

-add *cell_names*

Adds the specified cells to the list of cells that are selected for verification. You can use wildcard characters to specify a range of cells.

-remove *cell_names*

Removes the specified cells from the list of cells that are selected for verification You can use wildcard characters to specify a range of cells.

-clear

Clears cells from the list of cells that are selected for verification.

-file *file_name*

Adds cells from the specfied file to the list of cells that are selected for verification. The file must contain cell names that are separated by spaces. You can use wildcard characters to specify the file name.

cell_names

Specifies cells for verification. You can use wildcard characters to specify a range of cells. This option clears the existing list of cells that are selected for verification.

DESCRIPTION

This command manages the list of cells that are selected for verification. By default, Formality verifies only the cells that match in the reference and implementation containers. Use this command only after the implementation cells are read.

EXAMPLES

This example clears the existing list of cells and selects the specified cell list for verification.

```
fm_shell> select_cell_list My*
Adding cell "Myincand" for verification
Adding cell "Myand" for verification
Adding cell "Mynand" for verification
Adding cell "Myor" for verification
Adding cell "Mynor" for verification
```

This example removes the specified cell list from the list of cells selected for verification.

```
fm_shell> select_cell_list -remove Mynand
Removing cell "Mynand" from verification
```

This example adds the specified cell list to the list of cells selected for verification.

```
fm_shell> select_cell_list -add Mynand
Adding cell "Mynand" for verification
```

SEE ALSO

```
report_cell_list(2)
debug_library_cell(2)
```

set_app_var

Sets the value of an application variable.

SYNTAX

```
string set_app_var
   -default
   var
   value
```

Data Types

```
var        string
value      string
```

ARGUMENTS

-default

Resets the variable to its default value.

var

Specifies the application variable to set.

value

Specifies the value to which the variable is to be set.

DESCRIPTION

The **set_app_var** command sets the specified application variable. This command sets the variable to its default value or to a new value you specify.

This command returns the new value of the variable if setting the variable was successful. If the application variable could not be set, then an error is returned indicating the reason for the failure.

Reasons for failure include:

- The specified variable name is not an application variable, unless the application variable **sh_allow_tcl_with_set_app_var** is set to true See the **sh_allow_tcl_with_set_app_var** man page for details.
- The specified application variable is read only.
- The value specified is not a legal value for this application variable

EXAMPLES

The following example attempts to set a read-only application variable:

```
prompt> set_app_var synopsys_root /tmp
Error: can't set "synopsys_root": variable is read-only
       Use error_info for more info. (CMD-013)
```

In this example, the application variable name is entered incorrectly, which generates an error message:

```
prompt> set_app_var sh_enabel_page_mode 1
Error: "sh_enabel_page_mode" is not an application variable
        Use error_info for more info. (CMD-013)
```

This example shows the variable name entered correctly:

```
prompt> set_app_var sh_enable_page_mode 1
1
```

This example resets the variable to its default value:

```
prompt> set_app_var sh_enable_page_mode -default
0
```

SEE ALSO

```
get_app_var(2)
report_app_var(2)
write_app_var(2)
```

set_architecture

Sets the architecture for a specific type of multiplier or a DesignWare multiplier instance.

SYNTAX

```
int set_architecture
   object_ID
   architecture_type
```

Data Types

```
object_ID string
architecture_type string
```

Enabled Shell Modes

Setup

ARGUMENTS

object_ID

Specifies the object ID of the multiplier.

architecture_type

Specifies the type of architecture to use for the multiplier. Select one of the following architecture types - carry-save array, non-Booth Wallace tree, or Booth-encoded Wallace tree for the architecture type argument:

- **csa** - Specifies carry-save array
- **nbw** - Specifies non-Booth Wallace tree
- **wall** - Specifies Booth-encoded Wallace tree

DESCRIPTION

This command sets the architecture for a specific type of multiplier when the multiplier architecture generation is enabled by setting the **enable_multiplier_generation** variable to *true*. The architectures correspond to the **csa**, **nbw** and **wall** multiplier architectures. The architecture types are carry-save array, non-Booth Wallace tree, and Booth-encoded Wallace tree, respectively.

EXAMPLES

The following command sets the carry-save array architecture for the multiplier identified by **r:/WORK/test/mult28**

```
fm_shell> set_architecture r:/WORK/test4/mul_28 csa
```

SEE ALSO

```
report_architecture(2)
architecture_selection_precedence(3)
dw_foundation_threshold(3)
enable_multiplier_generation(3)
hdlin_multiplier_architecture(3)
```

set_black_box

Marks the specified designs as black boxes.

SYNTAX

```
status set_black_box
   [-attribute attribute_string]
   [designId_list]
```

Data Types

```
attribute_string string
designId_list string
```

Enabled Shell Modes

Setup

ARGUMENTS

-attribute *attribute_string*

Marks the designs that have the specified attribute as a black box.

designId_list

Marks the specific designs or instance cells as a black box.

DESCRIPTION

This command marks specific designs or instance cells as black boxes, which affects the verification of those designs. The black boxes are persistent during the current session.

To generate a black box for a specific design or cell, specify the *designID_list* option.

EXAMPLES

The following example marks the design *lower* as a black box.

```
fm_shell (setup)> set_black_box ref:/WORK/lower
Set black box on 'ref:/WORK/lower'
1
```

The following example marks the cell *inst1* of top module *top* as a black box.

```
fm_shell (setup)> set_black_box ref:/WORK/top/inst1
Set black box on 'ref:/WORK/top/inst1'
1
```

SEE ALSO

```
remove_black_box(2)
report_black_boxes(2)
```

set_cell_type

Sets a cell type value on a technology library cell or a collection of tech-lib cells.

SYNTAX

```
status set_cell_type
   [-value value_string]
   [designId_list]
```

Data Types

```
value_string string
designId_list string
```

Enabled Shell Modes

Setup

ARGUMENTS

-value value_string

Specifies the value of the cell type that needs to be added.

designId_list

Specifies a tech-lib cell on which the value of the cell type needs to be added. It is the string name of exactly one tech-lib cell or a collection of one or more cells.

DESCRIPTION

Sets the cell type on a technology library cell. You can specify the string name of exactly one tech-lib cell or a collection of one or more cells. The types that can be added are lssd: Level-Sensitive Scan Design retention: Retention register synchronizer: Synchronizer cell multibit: Banked register or Multi-bit register.

EXAMPLES

The following example sets the design *lower* with cell-type value LSSD.

```
fm_shell (setup)> set_cell_type -value lssd ref:/WORK/lower
LSSD is set on r:/WORK/lower
1
```

The following example sets the design *lower* with cell-type value RETENTION.

```
fm_shell (setup)> set_cell_type -value retention ref:/WORK/lower
RETENTION is set on r:/WORK/lower
1
```

SEE ALSO

```
remove_cell_type(2)
report_cell_type(2)
```

set_clock

Sets the specified net as a clock.

SYNTAX

```
set_clock
   netID
```

Data Types

```
netID string
```

Enabled Shell Modes

Setup

ARGUMENTS

netID

Sets the specified design net as a clock. If you specify a name consisting of a regular expression that resolves to more than one net, the operation is applied to all the matching nets.

DESCRIPTION

This command sets the specified net as a clock. Use this command to identify clock-gate latches in the fanin of compare points that the tool cannot identify as clocks (primary output ports and black box input pins), when the **verification_clock_gate_hold_mode** variable is set to *collapse_all_cg_cells*.

EXAMPLES

The following example sets the net **$impl/SCLK** as a clock.

```
fm_shell (setup)> set_clock $impl/SCLK
Set clock at 'OPT:/WORK/CORE/SCLK'
1
```

SEE ALSO

```
remove_clock(2)
report_clocks(2)
```

set_compare_point

This command is replaced by the **set_user_match** command.

DESCRIPTION

This command is replaced by the **set_user_match** command.

SEE ALSO

```
set_user_match(2)
```

set_compare_rule

Defines a name matching rule that Formality applies to a design to create compare points.

SYNTAX

```
status set_compare_rule
   [-type type]
   [-from expression]
   [-to string]
   [-file file_name]
   [designID]
```

Data Types

```
type string
expression string
string string
file_name string
designID string
```

ARGUMENTS

-type *type*

Specifies the design object type. Specify one of the following object types:

- *port*
- *net*
- *cell*

-from *expression*

Specifies the pattern to search for when setting the compare points.

-to *string*

Specifies the replacement pattern to use after successfully finding the search pattern. Arithmetic expressions can be specified in the replacement pattern delimited by \\(and \\).

-file file_name

Specifies a file containing one or more name matching rules.

designID

Specifies the design affected by the name matching rule.

DESCRIPTION

Use this command to define name translation rules. Formality uses these rules when creating compare points. This command is useful when similar design object names in two design use different naming formats. For example, the reference design uses a naming scheme where an underscore character is part of the register name:

```
reg_1
reg_2
reg_3
```

The implementation design uses a naming scheme where the bit numbers follow the bus name:

```
reg[1]
reg[2]
reg[3]
```

Use the **set_compare_rule** command to define a name translation rule from one format to another. Formality applies this rule when creating the compare points.

Compare rules are based on a regular expression syntax similar to those supported by the UNIX **sed** command. Arithmetic expressions are supported in the replacement pattern to handle vector name transformations introduced by bit-reversal, bit-renumbering, and partitioning. Operators + - * / and % are supported in arithmetic expressions. You can use the **test_compare_rule** command to ensure that your compare rules behave as required.

To apply several rules at the same time, use the *-file* option and specify a file that contains the rules. The contents of the file must include a single *search_pattern* and a single *replace_pattern* per line. Specify the *from* pattern first and then the *to* pattern followed by an optional *type*. Separate the patterns with a space. Formality reads the rules and applies each during the compare point creation process.

Compare rules remain in effect for the particular design until you remove the rule. To remove the rule, use the **remove_compare_rules** command.

If you do not specify a design ID, the tool applies the rules to the current design. If you do not specify a container name when specifying the design ID, Formality applies the compare rule to the design in the current container.

You can specify a particular design object type to be affected by specifying the **-type** *type* option. If you do not specify the *type* argument with the *-type* option, the tool applies the rules to all design objects.

EXAMPLES

This example defines a compare rule for the *async.reset.db* design. Primary output ports driven by a bus in this design use a bus_reg[bit] format. However, the design that is being verified uses the A_bus_reg_bit format for the same output ports. Before creating compare points, the tool applies the command rules and transforms the bus_reg[bit] name format to the A_bus_reg_bit name format.

```
fm_shell> set_compare_rule async.reset.db -from {\(.*_reg\)_\[\([0-9]*\)\]} -to {A_\1_\2}
```

This example defines several rules using the file *my_rules*. This example affects the default design in the current container.

```
fm_shell> set_compare_rule  -file my_rules
```

The following example defines a compare rule with arithmetic operators. The rule is expected to transform the following name formats:

```
 out0     ->    out_1_reg
 out1     ->    out_2_reg
 out2     ->    out_3_reg
 out3     ->    out_4_reg
```

```
fm_shell> set_compare_rule r:/WORK/topxor -from {out\([0-9]+\)} -to {out_\(\1+1\)_reg}
```

SEE ALSO

```
remove_compare_rules(2)
report_compare_rules(2)
test_compare_rule(2)
```

set_constant

Sets a net, port, cell, or pin to a constant state of 0 or 1.

SYNTAX

```
status set_constant
   [-type type]
   objectID
   constant_value
```

Data Types

```
type string
objectID string
constant_value string
```

Enabled Shell Modes

Setup

ARGUMENTS

-type *type*

Specifies the type of the design object. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following object types:

- *pin* - Specify a pin that is not a primitive input pin.
- *port*
- *net*
- *cell* - Specify a cell that is a register.

objectID

Specifies the design object. If you specify a name that resolves to more than one object, the name is applied to all of the matching objects. If the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects are affected in the following precedence: pin, port, net, cell. The exception is, if the design object, you are assigning the constant to, is a port connected to a net of the same name.

constant_value

Specifies the logic state:

- 0: A constant whose value is 0
- 1: A constant whose value is 1

DESCRIPTION

This command applies the logic 0 or 1 value to the specified design objects. Applying the constants to a circuit is helpful when problem areas exist in a circuit and you want to isolate the area by disabling an area of logic. For example, when your implementation design has scan logic and you want to exclude it from the verification process, you can assign a constant to the scan-enable input port to disable the scan logic from the verification process. If you assign a constant to a net with a driver, the tool issues a warning message.

Use the **report_constants** command to list the user-specified constants.

EXAMPLES

The following example sets port CC to a logical 1 state.

```
fm_shell (setup)> set_constant ref:/WORK/CORE/CC 1 -type port
Set 'ref:/WORK/CORE/CC' to constant 1
1
```

SEE ALSO

```
remove_constant(2)
report_constants(2)
```

set_constraint

Creates an external constraint on a design or hierarchal design.

SYNTAX

```
set_constraint
   [-name constraint_name]
   [-map mapping_list]
   constraint_type
   control_point_list
   [designID]
```

Data Types

```
constraint_name string
mapping_list string
constraint_type string
control_point_list string
designID string
```

ARGUMENTS

constraint_type

Specifies the constraint type. Specify one of the following values:

- *0hot* - Specifies a zero-hot constraint. Elements in the *control_point_list* are constrained so that all valid vectors have one of the control point elements at a logic 0 state with the remainder of the control points at logic 1.
- *0hot_init0* - Same as 0hot except that the control point elements are allowed to be in the all-logic 0 state.
- *0hot_init1* - Same as 0hot except that the control point elements are allowed to be in the all-logic 1 state.
- *1hot* - Specifies a one-hot constraint. Elements in the *control_point_list* are constrained so that all valid vectors have one of the control point elements at a logic 1 state with the remainder in the all-logic 0 state.
- *1hot_init0* - Same as 1hot except that the control point elements are allowed to be in the all-logic

0 state.

- *1hot_init1* - Same as 1hot except that the control point elements are allowed to be in the all logic 1 state.
- *coupled* - Specifies a coupled constraint. The elements in the *control_point_list* are always in the same binary state.
- *mutex* - Specifies a mutually exclusive constraint. The elements in the *control_point_list* are always in opposite binary states. The *control_point_list* must contain two elements for this type.
- *user_defined_type_name* - Specifies a user-defined constraint that is created by the **create_constraint_type** command. Use the name referenced by **create_constraint_type**.

control_point_list

Lists control points (primary inputs, registers, nets) in the constrained design or hierarchy to which to apply the constraint. When specifying the *control_point_list*, use the following syntax:

```
[ {] name [name] [}]
```

Where the *name* argument names a state holding flip-flop, port, or net in the constrained design or hierarchy. If name is not a full path name then it is relative to the given design or the current design if the design is not given. Use { } when the list contains more than one name.

designID

Specifies the design to which the tool resolves relative control point names.

-name *constraint_name*

Specifies the constraint's name.

-map *mapping_list*

Lists port names in the constraint module and the corresponding flip-flop, port, or net (control point) names in the constrained design. If you don't specify a mapping list, the tool implies mapping in which the names of the constraint module ports are assumed to be the same as the names of the corresponding control points in the constrained design. This option is only used in conjunction with a user-defined constraint type. You only need to specify a map for those constraint module ports that will not match by name. When specifying the mapping list, use the following syntax:

```
[ {] cm_name=name} [cm_name=name] [}]
```

Where:

- The *cm_name* argument specifies the name of an input port of the constraint module specified when the constraint module is created.
- The *name* argument specifies the leaf/simple name of a state-holding flip-flop, port, or net control point. As a shorthand, a control point can be referenced by position in the control point list using the syntax '%n' where n is the position in the list. For example $1 refers to the first control point.

Use the outer braces { } when the list contains more than one set of mapped names or the $n shorthand reference is used.

DESCRIPTION

This command applies an external constraint to the listed flip-flops, ports, and nets in the specified design or hierarchy.

The *type_name* argument specifies how the control points are to be constrained. If you don't specify a design, relative names are resolved to the current design. The type name can be either a predefined constraint type or a constraint type that you create using the **create_constraint_type** command. The predefined constraint types are:

- *0hot*
- *0hot_init0*
- *0hot_init1*
- *1hot*
- *1hot_init0*
- *1hot_init1*
- *coupled*
- *mutex*

When a constraint is applied to a design or hierarchy, all subsequent verifications verify equivalence only for conditions (vectors) that are legal according to the constraint. For example, if a 1hot constraint is applied to a set of ports, the tool does not check for equivalence in cases where the ports are not in a valid one-hot state.

If all control points are found in a single design, then all instances of that design within the verification have the same constraint. If the constraints are applied to the reference design, they are also implicitly applied to the comparable design in the implementation design, and vice versa.

Link the constrained container before executing this command. The tool reports an error if you apply a constraint on a flip-flop cell that is currently a black box because it is still unlinked or unsuccessfully linked.

When you execute this command, the tool assigns a unique name to the constraint, either automatically or explicitly using the *-name constraint_name* option. The **remove_constraint** and **report_constraint** commands require the unique constraint name.

Defining a constraint on a design input port is meaningful only for the top-level design because the instantiating design defines the constraints for lower-level designs. A warning occurs during verification if you apply a constraint to an input port of a design that is not the top-level design. In this case, the constraint is disabled for the higher-level verifications. In addition, the tool reports an error if you define a constraint for an output port of a design that is not a black box.

The **set_constraint** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example assigns a 1hot constraint to a set of registers in a design.

```
fm_shell> set_constraint 1hot {Q_reg[0] Q_reg[1] Q_reg[2]} ref:/WORK/testcase
FM_CONSTRAINT_0
```

The following example assigns a user-defined constraint to a mixed set of registers and ports in the reference design. The ports of the constraint module are explicitly mapped to control points in the constrained design hierarchy.

```
fm_shell> set_reference ref:/WORK/testcase
Reference design set to 'ref:/WORK/testcase'
1
fm_shell> create_constraint_type 2hot type:/WORK/my_2hot
1
fm_shell> set_constraint 2hot {PortData[0] U0/QR_reg[0] PortData[1] U1/QR_reg[0]}
    -map {IN1=PortData[0] IN2=$2 IN3=PortData[1] IN4=$4} \
-name MY2HOT ${ref}
MY2HOT
```

SEE ALSO

```
create_constraint_type(2)
remove_constraint(2)
remove_constraint_type(2)
report_constraint(2)
report_constraint_type(2)
```

set_current_command_mode

SYNTAX

```
string set_current_command_mode


   -mode command_mode | -command command

string command_mode
string command
```

ARGUMENTS

-mode *command_mode*

Specifies the name of the new command mode to be made current. **-mode** and **-command** are mutually exclusive; you must specify one, but not both.

-command *command*

Specifies the name of the command whose associated command mode is to be made current. **-mode** and **-command** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

The **set_current_command_mode** sets the current command mode. If there is a current mode in effect, the current mode is first canceled, causing the associated clean up to be executed. The initialization for the new command mode is then executed as it is made current.

If the name of the given command mode is empty, the current command mode is cancelled and no new command mode is made current.

A current command mode stays in effect until it is displaced by a new command mode or until it is cleared by an empty command mode.

If the command completes successfully, the new current command mode name is returned. On failure, the command returns the previous command mode name which will remain current and prints an error message unless error messages are suppressed.

EXAMPLES

The following example sets the current command mode to a mode called "mode_1"

```
shell> set_current_command_mode -mode mode_1
```

The following example clears the current command mode without setting a new mode.

```
shell> set_current_command_mode -mode ""
```

The following example sets the current command mode to the mode associated with the command "modal_command"

```
shell> set_current_command_mode -command modal_command
```

set_cutpoint

Specifies a hierarchical pin or net as a hard cutpoint.

SYNTAX

```
set_cutpoint
   [-type type]
   [-primary_input]
   objectID
```

Data Types

```
type string
objectID string
```

Enabled Shell Modes

Setup

ARGUMENTS

-type *type*

Specifies the object type when using the *objectID* argument. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following for the type argument:

- *pin*
- *net*

-primary_input

Specifies that the cutpoint is treated like a primary input. Constraints are not propagated through the cutpoint and it is constrained to 0 or 1 for verification.

objectID

Sets the design object as a cutpoint. If you specify a name that resolves to more than one object, the operation is applied to all matching objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, either pins or nets with the specified name are set as cutpoints.

DESCRIPTION

This command sets the specified hierarchical pin or net as a hard cutpoint. They are verified independently and are used as a free variable for verification of downstream compare points.

EXAMPLES

The following example uses the **-type net** option to set net $impl/PC[1], as a cutpoint.

```
fm_shell (setup)> set_cutpoint -type net $impl/PC[1]
Set cutpoint at 'OPT:/WORK/CORE/PC[1]'
1
```

The following example sets all of the pins on a block as cutpoints using the **-type pin** option.

```
fm_shell (setup)> set_cutpoint -type pin HDL:/WORK/CORE/MUX_OUT_BLK/*
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/DATA[10]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/DATA[11]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/DATA[12]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[1]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[2]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[3]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/STK_DATA[4]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[1]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[2]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[3]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[4]'
Set cutpoint at 'HDL:/WORK/CORE/MUX_OUT_BLK/UPC_DATA[5]'
1
```

SEE ALSO

```
remove_cutpoint(2)
report_cutpoints(2)
set_dont_cut(2)
create_cutpoint_blackbox(2)
```

set_direction

Sets the direction of ports and pins.

SYNTAX

```
status set_direction
   [-type object_type]
   [-shared_lib]
   objectID
   new_direction
```

Data Types

```
object_type string
objectID string
new_direction string
```

Enabled Shell Modes

Setup

ARGUMENTS

-type *object_type*

Specifies the type for the *object_type* argument. Use this option if the name of the specified design object is associated with more than one type of design object in the same design. If you specify a design object that has the same name as other design objects of different types, use the *-type* option to differentiate it from the other objects.

Specify one of the following values for the *object_type* argument:

- *port* to specify a port type
- *pin* to specify a pin type

-shared_lib

Redirects objects from technology libraries.

objectID

Specifies a design object. You can specify any unlinked port or pin.

new_direction

Specifies the port or pin direction. Specify one of the following directions:

- *in*
- *out*
- *inout*

DESCRIPTION

This command sets the direction on a specified unlinked port or pin.

Defining the direction of a pin or port helps determine equivalence more accurately during verification. For example, designs that use black boxes to hold the most recent value from a bus (bus-holder or bus-keeper) might not compare to designs that do not use comparable black boxes. For example, if the pin connecting the bus-holder to the bus is bidirectional, the bus might have an external driver.

EXAMPLES

In the following example, the direction of the pin connecting the black box to the bus is bidirectional. The following command redefines the pin's direction as input only. In this case, redefining the direction eliminates an extraneous driver from the bus.

```
fm_shell (setup)> set_direction impl:/WORK/CORE/CC in
Set 'ref:/WORK/CORE/CC' to direction in
1
```

SEE ALSO

set_dont_cut

Specifies an object that should not be used as a cutpoint.

SYNTAX

```
set_dont_cut
   objectID
   [-type objectID_type]
```

Data Types

```
objectID string
objectID_type string
```

Enabled Shell Modes

Setup

ARGUMENTS

-type *objectID_type*

Specifies the type of the specified object. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following object types:

- *pin*
- *net*

objectID

Specifies the design object that is not to be used as a cutpoint. If you specify a name consisting of a regular expression that resolves to more than one object, the operation is applied to all the matching objects. If the name specified resolves to multiple objects with identical names, and you do not specify the object type, only one of these objects will be affected. The precedence in this case is pin, net.

Wildcards may be used to specify objects.

DESCRIPTION

This command specifies a hierarchical pin or net that should not be a cutpoint. The specified object is not verified independently and is not used as a free variable for the verification of compare points.

The Formality tool does not insert automatic cutpoints at user-specified dont_cut points. The **set_dont_cut** command overrides the **set_cutpoint** command. Use this command if automatically inserted cutpoints (type PDCut) appear to be causing failing points.

EXAMPLES

The following example shows how to use the **set_dont_cut** command.

```
fm_shell (setup)> set_dont_cut $ref/block/pin
Set dont_cut at 'r:/WORK/top/block/pin'
1
```

SEE ALSO

remove_dont_cut(2)
report_dont_cuts(2)
set_cutpoint(2)
verification_insert_upf_isolation_cutpoints(3)

set_dont_match_points

Specifies a list of compare points to exclude from match.

SYNTAX

```
status set_dont_match_points
   [-type type]
   objectID_list
```

Data Types

```
type string
objectID_list string
```

ARGUMENTS

-type *type*

Specifies the object type. Use this option if the name of the specified design object is associated with more than one object type in the design. Specify one of the following values for the *type* argument:

- *pin*
- *port*
- *net*
- *cell*

objectIDlist

Specifies a list of objects to exclude from matching. The dont_match attribute is set on the specified objects. If the specified regular expression resolves to more than one object, the attribute is applied to all he objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the following precedence: pin, port, net, cell.

DESCRIPTION

This command applies the dont_match attribute on the specified objects. All objects with the dont_match attribute are excluded from matching.

The specified objects are excluded from matching during the subsequent *match* or *verify* command. To report the objects that have the dont_match attribute, use the **report_dont_match_points** command.

EXAMPLES

The following example shows how to prevent matching an object in the current design.

```
fm_shell (setup)> set_dont_match_points buffer_test
```

The following example shows how to prevent matching the specified test ports in the reference design named top.

```
fm_shell (setup)> set_dont_match_points {r:/WORK/top/test_so1 r:/WORK/top/test_so2
          r:/WORK/top/test_so3}
```

The following example shows how to use a wildcard character to prevent matching of test input ports on a black box.

```
fm_shell (setup)> set_dont_match_points {r:/WORK/top/ram32x20m/test_si*}
```

SEE ALSO

```
remove_dont_match_points(2)
report_dont_match_points(2)
report_matched_points(2)
report_unmatched_points(2)
```

set_dont_verify_points

Prevents design equivalence verification between two objects that constitute a matched compare point.

SYNTAX

```
status set_dont_verify_points
   [-type type]
   -directly_undriven_output
   [-propagate]
   objectID_list
```

Data Types

```
type string
objectID_list string
```

ARGUMENTS

-type *type*

Specifies the object type argument for the option. Use this option if the name of the specified design object is associated with more than one object type in the design. Specify one of the following values for the *type* argument:

- *pin*
- *port*
- *net*
- *cell*

The only compare points of type pin are those at black box input pins. Therefore, when you specify type pin, the tool disables the specified compare points at black box input pins.

-directly_undriven_output

Sets the dont_verify attribute on the top-level reference design output ports that do not have connected nets and on the connected nets that have no driving pins.

-propagate

Sets the dont_verify attribute on upstream compare points that are not read by other compare points. Use this option on primary output ports that are driven by test inserted lockup latches and cause failures on the ports.

objectIDlist

Sets the dont_verify attribute on design objects that are associated with the specified compare point. If the specified regular expression resolves to more than one object, the operation is applied to all of the matching objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the following precedence: pin, port, net, cell. Use this option to disable an unmatched compare point.

DESCRIPTION

This command applies the dont_verify attribute on the specified compare points. Compare points with the dont_verify attribute are excluded from verification.

The dont_verify points are not applied to the designs until you run the *match* or *verify* command. To report the compare points that have the dont_verify attribute, use the **report_matched_points** *-point_type* and the **report_unmatched_points** *-point_type dont_verify* commands.

EXAMPLES

The following example disables verification of an unmatched compare point in the current design.

```
fm_shell (setup)> set_dont_verify_points buffer_test
```

The following example disables the test ports in the reference design named top.

```
fm_shell (setup)> set_dont_verify_points {r:/WORK/top/test_so1
       r:/WORK/top/test_so2 r:/WORK/top/test_so3}
```

The following example uses a wildcard character to disable all of the test input ports on a black box cell because scan reordering has occurred in the implementation and the function at those inputs is known to be different from the reference.

```
fm_shell (setup)> set_dont_verify_points {r:/WORK/top/ram32x20m/test_si*}
```

SEE ALSO

```
remove_dont_verify_points(2)
```

```
    report_aborted_points(2)
    report_dont_verify_points(2)
    report_failing_points(2)
    report_matched_points(2)
    report_passing_points(2)
    report_unmatched_points(2)
```

set_dp_int_round

Specifies the initial rounding information for multipliers.

SYNTAX

```
set_dp_int_round
   objectID
   ExtPos [ IntPos ]
```

Data Types

```
   objectID string
   ExtPos int
   intPos int
```

ENABLED SHELL MODES

Setup

ARGUMENTS

objectID

Specifies the design multiplier to be rounded.

ExtPos IntPos

Specifies the external and internal rounding positions.

DESCRIPTION

Use this command to specify the internal and external rounding modifications applied to a multiplier.

EXAMPLES

```
fm_shell (guide)> set_dp_int_round { mult_12* mult_34* } 12 4
1
```

SEE ALSO

```
guide_set_rounding(2)
remove_dp_int_round(2)
report_dp_int_round(2)
```

set_eco_implementation

Specifies the container that contains the ECO implementation design.

SYNTAX

```
status set_eco_implementation
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

DESCRIPTION

This command specifies the container that contains the ECO implementation design.

SEE ALSO

```
match_eco_regions
write_eco_regions
create_eco_patch
set_orig_reference
set_orig_implementation
set_eco_reference
```

set_eco_reference

Specifies the container of the ECO reference design.

SYNTAX

```
status set_eco_reference
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

DESCRIPTION

This command specifies the container of the ECO reference design.

SEE ALSO

```
match_eco_regions
write_eco_regions
create_eco_patch
set_orig_reference
set_orig_implementation
set_eco_implementation
```

set_factor_point

Specifies that the given object is a factoring variable.

SYNTAX

```
set_factor_point
   [-type type]
   objectID_list
```

Data Types

```
type string
objectID_list string
```

Enabled Shell Modes

Setup
Match
Verify

ARGUMENTS

-type *type*

Specifies the type of the object. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following for the type argument:

- *port*
- *pin*
- *net*
- *cell*

objectID_list

Specifies a list of design objects to be designated as factoring variables. If you specify a name consisting that resolves to more than one object, the operation is applied to all the matching objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the following precedence: pin, port, net, cell.

DESCRIPTION

This command specifies that the list of given objects are factoring variables. Each object must be a primary input port, black box output pin, register cell or previously set cutpoint, and should be matched and binary constrained. Use wildcard characters to specify groups of objects.

User-specified factoring variables are not used until the *match* or *verify* commands. Any factoring variable that is not matched and binary constrained at the beginning of the subsequent verification is ignored. The tool performs factored verification using any remaining valid factoring variables. If no valid factoring variables are specified, then the tool performs unfactored verification as usual.

EXAMPLES

The following example sets the factors point on a design object state1_reg.

```
fm_shell (setup)> set_factor_point $ref/M1/U1/state1_reg
Set factoring variable at 'r:/WORK/top/M1/U1/state1_reg'
1
```

SEE ALSO

```
remove_factor_point(2)
report_factor_points(2)
```

set_fsm_encoding

Specifies the bit encodings for states in the specified design.

SYNTAX

```
status set_fsm_encoding
   [-name fsm_name]
   encoding_list
   [designID]
```

Data Types

```
fsm_name string
encoding_list string
designID string
```

Enabled Shell Modes

Setup

ARGUMENTS

-name *fsm_name*

Specifies the name of the FSM to which the tool applies the encoding. If a design has multiple FSMs, specify a FSM using the *-name fsm_name* option.

encoding_list

Specifies a list of all states in the FSM paired with assigned bit encodings. The syntax to specify the encoding list is:

```
[{] name=[number_base]code [name=[number_base]code] [}]
```

Use braces around the entire list when the list has more than one name-state pairs, or when you include spaces within a pair. If you include spaces as part of a name-state pair, enclose the pair in double quotation marks for an established FSM state vector flip-flop name or an optional number base. You can use two forms to specify the number base:

- 2# | 8# | 16# - Specifies binary, octal, or hexadecimal number bases, respectively. The decimal base is not supported.

- ^B | ^O | ^H - Specifies binary, octal, or hexadecimal number bases, respectively. The decimal base is not supported. This is the encoding associated with the FSM state vector flip-flop. Specify a string of digits in the base you specify.

designID

Specifies the design to which the tool applies the encoding.

DESCRIPTION

This command defines encodings for established FSM state vector flip-flops. Use this command to define encodings when you have just a few states to define. For FSMs with many states, use the **read_fsm_states** command.

You cannot use this command until you have established the FSM state vector flip-flop names. You can establish these names using the **set_fsm_state_vector** command.

If a design has only one FSM, you can define encodings in a specific design by specifying a design. Alternatively, you can let the tool assign encodings to the current design by not specifying the design.

If the tool cannot find the design, it returns the following error message,

```
Design ID name is invalid
```

EXAMPLES

The following example declares four states for two previously named flip-flops. In this example, the encodings are applied to the current design. Encodings are specified in the binary numbering system.

```
fm_shell (setup)> set_fsm_encoding { S0=2#00 S1=2#01 S2=2#10 S3=2#11 }
1
```

The following example accomplishes the same result as the previous example. However, states are supplied in the hexadecimal numbering system.

```
fm_shell (setup)> set_fsm_encoding { S0=16#0 S1=16#1 S2=16#2 S3=16#3 }
1
```

The following example defines encodings for a FSM design named *ref:/WORK/CORE/FSM*. This example specifies the state using the octal numbering system. All state encodings are enclosed in double quotation marks. In this example, spaces around the equal sign are not allowed.

```
fm_shell (setup)> set_fsm_encoding "STATE_1=8#1 STATE_2=8#2"
1
```

SEE ALSO

```
read_fsm_states(2)
report_fsm(2)
set_fsm_state_vector(2)
```

set_fsm_state_vector

Names state vector flip-flops in an FSM.

SYNTAX

```
string set_fsm_state_vector
   flip-flop_list
   [ designID ]
   [ -name FSM_name ]
   [ -match matching_fsm_name ]
```

Data Types

```
flip-flop_list string
designID string
FSM_name string
matching_fsm_name string
```

Enabled Shell Modes

Setup

ARGUMENTS

flip-flop_list

Specified the ordered list of flip-flops in the FSM used to store the state vector. Use the following syntax to specify the flip-flop list:

```
[ { ] name [ name... ] [ } ]
```

Use the [ ] when the list has more than one flip-flop. The *name* argument specifies the instance name of a state-holding flip-flop in the FSM.

designID

The design to which Formality applies the names.

-name *FSM_name*

Specifies the name of the FSM to which the state vector belongs.

-match fImatching_fsm_name

Specifies the FSM name mapping between the reference and implementation designs. This option is only needed when there are more than 1 re-encoded FSMs within the same level of hierarchy.

DESCRIPTION

This command names and establishes the order of each flip-flop used to store a state vector in an FSM. The number of flip-flops in the list also determines the length of the state encodings used to represent each state.

The list of state vector names you supply should map directly to the encodings for all states. The first element in the instance list stores the left-most bit (most significant bit) of each state encoding; the second element in the list stores the next bit of each state encoding; and so on. It is an error to have the same flip-flop in two FSMs.

For example, to encode four states using two bits with flip-flops named ff1 and ff0, specify the following command:

```
   fm_shell (setup)> set_fsm_state_vector { ff0 ff1 }
```

In this case, the set of all state encodings in binary is:

```
00
01
10
11
```

Instance ff0 stores the left column for the state encodings, and ff1 stores the right column. After you name the flip-flops, you can define their encodings by using the **set_fsm_encoding** command.

You can name and order flip-flops in a specific FSM by supplying a *designID*. You can also let Formality apply the list to the current design by omitting a *designID*.

If Formality cannot find a listed flip-flop in the design, it returns the following error message, where *name* is the flip-flop name you specified and *container* is the container:

```
Error: A cell named name could not be found in container container.
```

When you execute this command, Formality assigns a unique name to the FSM, either automatically or explicitly using the -name *FSM_name* option. This name is reported by **report_fsm**.

If there are multiple re-encoded FSMs within a single level of hierarchy, you need to use the **-match** option to map the reference and implementation FSM names to each other. First, use the **set_fsm_state_vector -name** command to specify the FSM names for the multiple FSMs that reside within the same level of hierarchy in the reference design. Then, when specifying the state vectors for the matching FSMs in the implementation design, use the **-match** option to map each FSM to the appropriate

FSM name in the reference design.

EXAMPLES

The following example names and orders three flip-flops in the current design. Because of the ordered nature of the list, subsequent encodings are applied such that FF_0 stores the first (left-most) bit, FF_1 stores the middle bit, and FF_2 stores the last (right-most) bit.

```
fm_shell (setup)> set_fsm_state_vector { FF_0 FF_1 FF_2 }
1
```

The following example names and orders the same list in an FSM design named *ref:/WORK/CORE/FSM_BEHAVIOR*. Since no *FSM_name* is provided, Formality creates a unique name for this FSM and returns the name.

```
fm_shell (setup)> set_fsm_state_vector { FF_0 FF_1 FF_2 } ref:/WORK/CORE/FSM_BEHAVIOR
FM_FSM_0
```

In the following example, the *FSM_name* is provided.

```
fm_shell (setup)> set_fsm_state_vector { FF_0 FF_1 FF_2 } ref:/WORK/CORE/FSM_BEHAVIOR -name my_fsm
my_fsm
```

SEE ALSO

```
read_fsm_states(2)
report_fsm(2)
set_fsm_encoding(2)
```

set_host_options

Sets options for parallel processing.

SYNTAX

```
status set_host_options
   [ -max_cores num_cores ]
   [ -local_process ]
```

Data Types

```
num_cores string
```

ARGUMENTS

-max_cores *num_cores*

Specifies the number of processes on the local machine to use in parallel. The default is one core. The maximum number of cores you can specify is eight. A second Formality license is consumed if you specify more than four cores. The second licensed is not released until Formality exits.

-local_process

Specifies that **-max_cores** applies to the local process. This is always true for now and may always be omitted.

DESCRIPTION

This command sets options for parallel processing.

EXAMPLES

This example specifies that up to 4 processes may be used on the local machine in parallel.

```
fm_shell> set_host_options -max_cores 4
```

SEE ALSO

set_implementation_design

Sets the implementation design.

SYNTAX

```
status set_implementation_design
   [designID]
```

Data Types

```
designID string
```

ARGUMENTS

designID

Specifies the design you want to establish as the top-level implementation design.

DESCRIPTION

This command establishes the implementation design. To establish a specific design as the implementation design, you must specify the *designID* argument. When you dont specifiy the *designID* argument, Formality sets the current design as the implementation design.

When you establish an implementation, it remains active until you issue a subsequent **set_implementation_design** command.

EXAMPLES

The following example establishes the design CORE in the impl container as the top-level implementation design.

```
fm_shell> set_implementation_design impl:/WORK/CORE
Implementation design set to 'impl:/WORK/CORE'
1
```

SEE ALSO

```
set_reference_design(2)
```

set_init_toggle_assumption

Sets an initial toggle assumption on a controlling object(s).

SYNTAX

```
set_init_toggle_assumption
   [-toggle | -no_toggle]
   [-type]
   [object_list]
```

Data Types

object_list string

Enabled Shell Modes

Setup

ARGUMENTS

-toggle

Specifies that the object(s) should toggle during initialization.

-no_toggle

Specifies that the object(s) should not toggle during initialization.

-type

Optional switch to specify the type of the object(s). The types are port, net or cell.

object_list

Specifies the object(s) whose initial toggle assumptions need to be set.

DESCRIPTION

Sets an initial toggle assumption on a controlling object(s).

EXAMPLES

The following example sets the object *i:/WORK/bit_slice/hclk* to toggle.

```
fm_shell (setup)> set_init_toggle_assumption -toggle i:/WORK/bit_slice/hclk
1
```

The following example sets the object *i:/WORK/bit_slice/hclk* to not toggle.

```
fm_shell (setup)> set_init_toggle_assumption -no_toggle i:/WORK/bit_slice/hclk
1
```

SEE ALSO

```
remove_init_toggle_assumption(2)
report_init_toggle_assumption(2)
```

set_input_value_range

Overrides the default binary (0 or 1) value range for a primary input. You can set the allowed value range to any subset of {0, 1, X, Z}.

SYNTAX

```
status set_input_value_range
   objectID
   value_range
```

Data Types

```
objectID string
value_range string
```

Enabled Shell Modes

Setup

ARGUMENTS

objectID

Specifies the design object whose value range you want to set. You can supply any PI port or pin. If you specify a name consisting of a regular expression that resolves to more than one object, the operation will be applied to all of the matching objects, with one exception: if the name you give resolves to multiple objects with identical names (and you do not specify the object type), only one of these objects will be affected; the precedence in this case is pin followed by port.

value_range

Specifies the value range as a subset of the standard 4-valued logic set {0, 1, X, Z}:

- *01X* - A set of values {0, 1, X}
- *0X* - A set of values {0, X}
- *1* - A constant whose value is 1

DESCRIPTION

This command sets the allowed value ranges for primary inputs. By default, all primary inputs are assumed to be binary (01), meaning they can take 0 or 1 value.

The **report_input_value_range** command is mainly used during hierarchical verification. This happens automatically when you use **write_hierarchical_verification_script** command. You may explicity set value ranges if additional constraints are known for an input.

Formality tracks user-defined input value ranges. Use the **report_input_value_range** command to view a list of these settings.

EXAMPLES

The following example sets port PI1 to input value range of 0, 1 or X.

```
fm_shell (setup)> set_input_value_range r:/WORK/top/PI1  01X
Set 'r:/WORK/top/PI1' to input range 01X
1
```

SEE ALSO

```
remove_input_value_range(2)
report_input_value_range(2)
write_hierarchical_verification_script(2)
```

set_inv_push

Adds a sequential object, top-level port, or black-box input pin to the list of objects through which Formality transports inverters for verification.

SYNTAX

```
status set_inv_push
   [ -shared_lib ]
   objectIDlist
```

Data Types

```
objectIDlist string
```

Enabled Shell Modes

Setup

ARGUMENTS

-shared_lib

Sets the inversion push on shared technology libraries.

objectIDlist

Specifies the list of objects to which you want to apply an inversion push. Use the following syntax when specifying the *objectID_list* argument:

```
[ { ] name [ name... ] [ } ]
```

Use { } when the list has more than one object. The *name* argument specifies the name of an object in the design.

DESCRIPTION

This command adds a sequential object, top-level output port or black-box input pin to the list of objects through which Formality transports inverters for successful verification.

You can only use this command on,

- `Sequential primitives`
- `Instances that reference a technology library`
- `Designs in technology libraries`
- `Top-level (primary) output ports`
- `Black-box input pins`

EXAMPLES

The following example defines a cell in which to apply inversion push.

```
fm_shell (setup)> set_inv_push ref:/WORK/CORE/reg_A
1
```

The following example defines a technology library in which to apply inversion push.

```
fm_shell (setup)> set_inv_push -shared_lib ref:/TECH_WORK/mydesign
1
```

SEE ALSO

```
remove_inv_push(2)
report_inv_push(2)
```

set_message_info

Set some information about diagnostic messages.

SYNTAX

```
string set_message_info -id message_id [-limit max_limit|-stop_on|-stop_ff]

string message_id
integer max_limit
```

ARGUMENTS

-id *message_id*

Information is to be set for the given *message_id*. The message must exist. Although different constraints allow different message types, no constraint allows severe or fatal messages.

-limit *max_limit*

Set the maximum number of occurrences for *message_id*. This is an integer greater than or equal to zero. If you set it to zero, that means the number of occurrences of the message is unlimited. Messages which occur after a limit is reached are automatically suppressed.

-stop_on

Force Tcl error if message is emitted.

-stop_off

Turn off a previous -stop_on directive

DESCRIPTION

The **set_message_info** command sets constraints on diagnostic messages (typically error, warning, and informational messages).

Currently, you can set a upper limit for the number of occurrences of a message. You can set this to zero to indicate that there is no limit. You can retrieve the current limit for a message using the **get_message_info** ***command. When the limit is exceeded, all future occurrences of the message are automatically suppressed. A count of total occurrences (including those suppressed) can be retrieved using get_message_info.***

EXAMPLES

The following example uses **set_message_info** to set a limit on the number of APP-027 messages to 100. When the 101st APP-027 message is about to be issued, you will be warned that the limit has been exceeded, and that all future occurrences will be suppressed.

```
prompt> set_message_info -id APP-027 -limit 100
prompt> do_command
Warning: can't find node U27.1 (APP-027)
Warning: can't find node U27.2 (APP-027)
Warning: can't find node U27.3 (APP-027)
 ...
Warning: can't find node U27.100 (APP-027)
Note - message 'APP-027' limit (100) exceeded.  Remainder will be suppressed.
1
```

SEE ALSO

```
get_message_info(2)
get_message_ids(2)
print_message_info(2)
suppress_message(2)
```

set_mismatch_message_filter

Sets warning or suppress filter on one or more simulation-synthesis mismatch messages.

SYNTAX

```
status set_mismatch_message_filter
   -warn | -suppress
   [-signal SignalName]
   [-block HierarchicalBlockName]
   [-file FileName]
   [-line LineNumber]
   [MismatchMessageIDList]
```

Data Types

```
   SignalName                string
   HierarchicalBlockName     string
   FileName                  string
   LineNumber                integer
   MismatchMessageIDList     list
```

ARGUMENTS

-warn

Reduces the mismatch message severity to warning. This option can not be combined with **-suppress**.

-suppress

Suppresses the mismatch message. This option can not be combined with **-warn**.

-signal SignalName

Sets mismatch message filter based on the Signal or Variable Name. The SignalName can accept a string value in Tcl glob style pattern. This is an optional option.

-block HierarchicalBlockName

Sets mismatch message filter based on the hierarchical block name. The Hierarchical block name can accept a string value in Tcl glob style pattern. The block name can be module or entity-architecture name, always or process block name, generate block name, function name and procedure name. The expected format of hierarchical block name for always or process block and generate block are as follows:

<ModuleName>[/<BlockName>]*

The expected format of hierarchical block name for functions and procedures that defined inside a module or package is as follows:

<ModuleOrPackageName>/<FunctionOrProcedureName>

This is an optional option.

`-file FileName`

Sets mismatch message filter based on a file name. The file name can be a leaf level file name like */test.v or full file path. This can accept values which are in TCL glob-style pattern matching form. This is an optional option.

`-line LineNumber`

Sets mismatch message filter for a mismatch message that occurs at given line number of an RTL file. This option requires **-file** option. It cannot be used with **-signal** or **-block** options. This is an optional option

`MismatchMessageIDList`

Sets mismatch message filter for the list of mismatch message Ids specified. This is an optional option. Below are the list of of simulation mismatch error codes that this option accepts:

FMR_VHDL-274 FMR_VHDL-1002 FMR_VHDL-1004 FMR_VHDL-1014 FMR_VHDL-1025 FMR_VHDL-1027 FMR_VHDL-1036 FMR_VHDL-1140 FMR_VHDL-1144 FMR_VHDL-1145 FMR_VLOG-079 FMR_VLOG-081 FMR_VLOG-083 FMR_VLOG-087 FMR_VLOG-089 FMR_VLOG-090 FMR_VLOG-091 FMR_VLOG-925 FMR_VLOG-928 FMR_VLOG-929 FMR_ELAB-034 FMR_ELAB-058 FMR_ELAB-059 FMR_ELAB-100 FMR_ELAB-115 FMR_ELAB-116 FMR_ELAB-117 FMR_ELAB-118 FMR_ELAB-125 FMR_ELAB-130 FMR_ELAB-136 FMR_ELAB-145 FMR_ELAB-146 FMR_ELAB-147 FMR_ELAB-149 FMR_ELAB-150 FMR_ELAB-151 FMR_ELAB-153 FMR_ELAB-154 FMR_ELAB-261

DESCRIPTION

The command sets the warning or suppress filter on one or more simulation-synthesis mismatch messages. This command can be used to either reduce the severity to warning or suppress the mismatch message based on the selection criteria given by the user. If suppressed, the message will not be displayed and does not report in the RTL Interpretation summary. Explicit filters are applicable only to those message Ids specified. An Implicit filter(a filter set without MismatchMessageIDList) is applicable for all mismatch messages.

Precedence of command options

From a given set of commands that act on same set of mismatch messages, the commands with most specific filter will have higher precedence. List of various combinations of options in decreasing order of precedence is given below:

-file + **-line** **-file** + **-block** + **-signal** **-file** + **-signal** **-file** + **-block** **-block** + **-signal** **-signal** **-block** **-**

file

If any two filters have same precedence as per above list, then the filter set with **-warn** will have higher priority over **-suppress** filter.

The command returns status 1 on success and 0 on failure.

EXAMPLES

To warn on all the mismatch messages, use the command

```
fm_shell (setup)> set_mismatch_message_filter -warn
```

To suppress all the FMR_ELAB-117, use the command

```
fm_shell (setup)> set_mismatch_message_filter -suppress FMR_ELAB-117
```

To suppress all the FMR_VLOG-079 that matches signal names start with set_, use the command

```
fm_shell (setup)> set_mismatch_message_filter -suppress -signal {set_*} FMR_VLOG-079
```

To warn on all the FMR_VLOG-079 in blocks under bot/gen0, use the command

```
fm_shell (setup)> set_mismatch_message_filter -warn -block {bot/gen0/*} FMR_VLOG-079
```

SEE ALSO

```
remove_mismatch_message_filter(2)
report_mismatch_message_filters(2)
```

set_net_resolution

Sets resolution function on the specified net.

SYNTAX

```
status set_net_resolution
   function
   objectID
```

Data Types

```
function      string
objectID      string
```

Enabled Shell Modes

Setup

ARGUMENTS

function

Specifies the function Formality uses for a net if it have multiple drivers. The supported function types are:

- *consensus* - Causes Formality to use the following rules to define the value of a multiply-driven net. This is the default net resolution function.
- If one or more drivers are on (for example, three state devices that have been enabled), and all these drivers have the same value, the net resolves to that value.
- If one or more drivers are on, and all these drivers do not have the same value, the net resolves to a don't care state (X).
- If none of the drivers are on, the net resolves to a high impendence state (Z).
- *blackbox* - Causes Formality to use an "unknown" function.
- *and* - Causes Formality to use a "wired-and" function. You should never use this resolution with

CMOS technology. Furthermore, your design must support wired-and functionality before you should use this resolution.

- *or* - Causes Formality to use a "wired-or" function. You should never use this resolution with CMOS technology. Furthermore, your design must support wired-or functionality before you should use this resolution.
- *onehot* - This resolution function is meant to be used when the reference design has an associated UPF file and the implementation design is a post P&R netlist with the power and ground network instantiated. It causes Formality to use the following rules to define the value of a multiply-driven net.
- If one and only one driver has value 1'b1 and all other drivers have value 1'b0, the net resolves to value 1'b1.
- If more than one driver has value 1'b1 the net resolves to undetermined state (X).
- If all the drivers have value 1'b0, the net resolves to value 1'b0.
- If any of the drivers have a non-binary value (X or Z), the net resolves to undetermined state (X).
- *parallel* - This resolution function is meant to be used when the reference design has an associated UPF file and the implementation design is a post P&R netlist with the power and ground network instantiated. It causes Formality to use the following rules to define the value of a multiply-driven net.
- If all the driver have value 1'b1, the net resolves to value 1'b1.
- If all the drivers have value 1'b0, the net resolves to value 1'b0.
- If some of the driver has value 1'b1 and others have value 1'b0, the net resolves to undetermined state (X).
- If any of the drivers have a non-binary value (X or Z), the net resolves to undetermined state (X).
- *parallel_onehot* - This resolution function is meant to be used when the reference design has an associated UPF file and the implementation design is a post P&R netlist with the power and ground network instantiated. Please refer to the UPF standard for more details.

objectID

Specifies the net on which the resolution function is to be applied. If you specify a name that resolves to more than one net, the resolution function is applied to all of the matching nets.

DESCRIPTION

Use this command to change the net resolution function of a net with multiple drivers.

EXAMPLES

The following examples sets parallel resolution function for the net VDD in the implementation design.

```
fm_shell (setup)> set_net_resolution \
parallel \
i:/WORK/dut/VDD
Set resolution 'parallel' on net 'i:/WORK/dut/VDD'
1
```

SEE ALSO

```
remove_net_resolution(2)
report_net_resolution(2)
```

set_orig_implementation

Specifies the container of the original implementation design.

SYNTAX

```
status set_orig_implementation
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

DESCRIPTION

This command specifies the container of the original implementation design.

SEE ALSO

```
match_eco_regions
write_eco_regions
create_eco_patch
set_orig_reference
set_eco_reference
set_eco_implementation
```

set_orig_reference

Specifies the container of the original reference design.

SYNTAX

```
status set_orig_reference
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

DESCRIPTION

This command specifies the container of the original reference design.

SEE ALSO

```
match_eco_regions
write_eco_regions
create_eco_patch
set_orig_implementation
set_eco_reference
set_eco_implementation
```

set_parameters

Sets the verification parameters for the specified design.

SYNTAX

```
status set_parameters
   [ -flatten ]
   [ -resolution function ]
   [ -retimed ]
   [ designID ]
```

Data Types

```
function string
designID string
```

Enabled Shell Modes

Setup (for the **-resolution** and **-retimed** options only)

ARGUMENTS

-flatten

Ignores hierarchical design block boundaries during verification. By default, Formality considers hierarchical design block boundaries to be potential internal cutpoints in the network to be verified. Although this behavior is not explicitly visible, under certain conditions, for example if you know a significant amount of boundary optimization has occurred in your design, you may see improved verification run-times by setting this parameter. However, for most designs, setting this parameter will result in a degradation or no change in verification run-time.

-resolution *function*

Specifies the function Formality uses for nets that have multiple drivers in the design. Formality uses "consensus" by default. You must supply one of the following values for function:

- *consensus* - Causes Formality to use the following rules to define the value of multiply-driven nets. This is the default net resolution function.
- If one or more drivers are on (for example, three state devices that have been enabled), and all these drivers have the same value, the net resolves to that value.
- If one or more drivers are on, and all these drivers do not have the same value, the net resolves to a don't care state (X).
- If none of the drivers are on, the net resolves to a high impendence state (Z).
- *blackbox* - Causes Formality to use an "unknown" function.
- *and* - Causes Formality to use a "wired-and" function. You should never use this resolution with CMOS technology. Furthermore, your design must support wired-and functionality before you should use this resolution.
- *or* - Causes Formality to use a "wired-or" function. You should never use this resolution with CMOS technology. Furthermore, your design must support wired-or functionality before you should use this resolution.

-retimed

Specifies that the design has been retimed, which means some registers inside the given design have been repositioned to meet a timing goal. As a result, Formality takes additional steps during verification to ensure successful compare point creation. This parameter is not to specify which register or registers have been repositioned, but to specify in which design there is at least one repositioned register. By default, Formality does not consider a design as having been retimed.

designID

Specifies the design for which you want to set parameters. For information about how to supply a *designID*, refer to the user guide.

DESCRIPTION

Use this command to set parameters that affect the verification of a specific design or the current design. To set parameters on a specific design, supply a *designID*. To apply parameters to the current design, omit the *designID*.

On a design-by-design basis, you can control hierarchical boundary consideration, resolution of nets with more than one driver, and the handling of retimed designs.

- *Flattening* - Given a top-level design that consists of hierarchical blocks, Formality considers hierarchical design block boundaries to be potential internal cutpoints in the network to be verified. Although this behavior is not explicitly visible, under certain conditions, for example if you know a significant amount of boundary optimization has occurred in your design, you may see improved verification run-times by setting this parameter. However, for most designs, setting this parameter will result in a degradation or no change in verification run-time. Setting the parameter on a given design instructs Formality to ignore the boundary of that block and all of its internal blocks.

- *Resolution* - By default, Formality uses the consensus rules as described in the -resolution switch to represent nets that have multiple drivers. You can use the -resolution switch to also specify "unknown", "wired-and", or "wired-or" functions to represent these nets. Using "unknown" (specifying black box) requires that the individual nets be treated as fixed points in the design and matched in an attempt to create compare points. Using "wired-and" or "wired-or" functions does not require the nets to be matched between the two designs in an attempt to create compare points.
- *Retiming* - Retimed designs can contain registers that have been moved in an attempt to optimize space or timing. This relocation can cause difficulties when Formality is creating compare points when given a design and its retimed counterpart. To accommodate retimed designs, you can use the -retimed switch. When you set the retimed parameter on a design, Formality takes steps to account for certain types of retiming. By default, Formality treats a design as if it has not been retimed.

EXAMPLES

The following example sets all three parameters on the CORE design. The command causes Formality to ignore hierarchical block boundaries in the hierarchical design. Formality also considers nets with multiple drivers as "wired-and" functions. By default, Formality applies a set of consensus rules to determine the nets functions. Finally, Formality takes steps to verify CORE against a retimed design.

```
fm_shell (setup)> set_parameters  -flatten \
-resolution AND \
-retimed \
ref:/WORK/CORE
Set parameter 'flatten' on 'ref:/WORK/CORE'
Set parameter 'retimed' on 'ref:/WORK/CORE'
Set resolution mode of 'AND' on 'ref:/WORK/CORE'
1
```

SEE ALSO

```
remove_parameters(2)
report_parameters(2)
```

set_probe_points

Sets a net pair between the reference and implementation designs for probe verification.

SYNTAX

```
set_probe_points
   [ -inverted ]
   ref_netID
   imp_netID
```

Data Types

```
   ref_netID string
   imp_netID string
```

ENABLED SHELL MODES

Setup Match Verify

ARGUMENTS

-inverted

Specifies that the reference and implementation net objects have an inverted relationship.

ref_netID

Specifies the reference net object that is a part of a probe pair to be compared for debugging. The name should specify the absolute path to a unique net in the reference design starting with the top design.

imp_netID

Specifies the implementation net object which is part of a probe pair to be compared for debugging. The name should specify the absolute path to a unique net in the implementation design starting with the top design.

DESCRIPTION

This command sets a probe pair for debug comparison. The probe pair consists of a net in the reference design and a second net from the implementation design. Nets that are otherwise compare points (such as cutpoints) cannot be set as probes. The net name should refer to a unique net. Use multiple **set_probe_point** commands to set multiple probe pairs with the same reference net and different implementation nets. This allows more efficient verification and debugging when attempting to find equivalent nets in a logic cone.

EXAMPLES

```
fm_shell (setup)> set_probe_points $ref/U1/U2/net2  $impl/U3/net4
Set user probe between 'r:/WORK/top/U1/U2/net2' and 'i:/WORK/top/U3/net4'
1
fm_shell (setup)> set_probe_points r:/WORK/top/M1/n1 i:/WORK/top/n1
Set user probe between 'r:/WORK/top/M1/n1' and 'i:/WORK/top/n1'
1
```

SEE ALSO

```
remove_probe_points(2)
report_probe_points(2)
report_probe_status(2)
verify(2)
```

set_reference_design

Sets the reference design.

SYNTAX

```
status set_reference_design
   [ designID ]
```

Data Types

```
designID string
```

ARGUMENTS

designID

Specifies the design to establish as the top-level reference design.

DESCRIPTION

This command establishes the reference design. To establish a specific design as the reference design, specify the *designID* argument. When you dont specify the *designID* argument, Formality sets the current design as the reference design.

When you establish a reference design, it remains active until you issue the next **set_reference_design** command.

EXAMPLES

The following example establishes the design CORE in the ref container as the top-level reference design.

```
fm_shell> set_reference_design ref:/WORK/CORE
Reference design set to 'ref:/WORK/CORE'
1
```

SEE ALSO

```
set_implementation_design(2)
```

set_run_alternate_strategies_options

Set alternate strategies job options.

SYNTAX

```
status set_run_alternate_strategies_options
   [-max_cores integer ]
   [-protocol LSF | SGE | RTDA | PBS | NB | CUSTOM ]
   [-submit_command command ]
   [-num_processes integer ]
```

Data Types

```
integer        integer
command        string
```

ARGUMENTS

-max_cores *integer*

Specifies the CPU core usage limit for alternate strategy workers.

-protocol LSF | SGE | RTDA | PBS | NB | CUSTOM

Specifies the communication protocol or farm type.

-submit_command *command*

Specifies the command for launching and submitting jobs to alternate strategy workers.

-num_processes *integer*

Specifies the maximum number of workers/strategies to start in parallel.

DESCRIPTION

This command configures the Master for running alternate strategy workers in parallel, and must be issued prior to run_alternate_strategies command.

Master Formality license is shared by one other alternate strategy worker. All other workers check out their own licenses, as and when required.

With this command, the following setup variables are deprecated and have no effect:

```
alternate_strategy_job_env
alternate_strategy_job_options
alternate_strategy_monitor_env
alternate_strategy_monitor_options
```

EXAMPLES

The following examples illustrate the command usage for configuring the master to run alternate strategies in parallel in different farm environments.

1. SGE farm - 16 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol SGE
   -submit_command "qsub -V -P bnormal -cwd
   -l arch=glinux,os_bit=64,mem_free=10G -pe mt 4" -num_processes 16
```

2. SGE farm - custom script for 16 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol SGE
   -submit_command custom.scr -num_processes 16

custom.scr
#!/bin/csh -f
qsub -V -P bnormal -cwd -l arch=glinux,os_bit=64,mem_free=10G -pe mt 4 $*
```

3. LSF farm - 8 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol LSF
   -submit_command "bsub -n 4 -R \"qscm\" -R \"rusage\[mem=32000\]\"
   -R \"span\[hosts=1\]\"" -num_processes 8
```

4. LSF farm - custom script for 8 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol LSF
   -submit_command custom.scr -num_processes 8

custom.scr
#!/bin/csh -f
bsub -n 4 -R "qscm" -R "rusage[mem=32000]" -R "span[hosts=1]" "$*"
```

5. RTDA - 12 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol RTDA
   -submit_command "nc run -r+ RAM/32000 -r+ CPUS/4" -num_processes 12
```

6. CUSTOM farm - 4 workers each using 4 cores

```
fm_shell> set_run_alternate_strategies_options -max_cores 4 -protocol CUSTOM
```

```
      -submit_command custom.scr -num_processes 4

   custom.scr
   #!/bin/csh -f
   bsub -n 4 -R "qscm" -R "rusage[mem=32000]" -R "span[hosts=1]" "$*"
```

SEE ALSO

```
run_alternate_strategies(2)
```

set_svf

Sets the automated setup file (.svf) for verification.

SYNTAX

```
status set_svf
[-append]
[-ordered]
[-extension name]
[filedirnames]
```

Data Types

```
name string
filedirnames string
```

Enabled Shell Modes

Setup
Guide

ARGUMENTS

-append

Appends the data to the current SVF information.

-ordered

Preserves the order of the file names when reading.

-extension *name*

Specifies the extension to be used when recursively searching directories for SVFs. The default is "svf".

filedirnames

Specifies the name of the SVF to read or directories to search. If nothing is specified, the SVF is reset.

DESCRIPTION

Use this command to set the automated setup file (.svf) for verification. The SVF provides valuable information that can be used during compare point matching to facilitate aligning of compare points in the designs to be verified. If multiple files are specified, by default, the Formality tool reads the files in the order of the file timestamps before processing. If you use the **-ordered** option, the tool does not change the order while reading the .svf files.

If a directory name is specified, Formality recursively searches the directory and reads any file whose extension matches with the **-extension** option or "svf" if the **-extension** option is not specified. For each SVF file found, the contents are processed and all information is stored in memory for later use during the name-based compare point matching.

If file name is not specified, the SVF is reset. However, it is preferred to use the **remove_guidance** command to delete the stored SVF data.

You can use the **set_svf** command only before reading a design. The Formality tool applies SVF processing to a specified reference design only once and ignores any additionally applied SVF during verification of a reference design that has already been modified by SVF processing.

You can see the contents of a previously set SVF with the **report_guidance -to filename** command.

EXAMPLES

This example sets the automated setup file for verification.

```
fm_shell (setup)> set_svf myfile.svf
SVF set to 'myfile.svf'.
1
```

SEE ALSO

```
report_guidance(2)
remove_guidance(2)
```

set_top

Resolves cell references and elaborates the RTL designs.

SYNTAX

```
status set_top
   [ -vhdl_arch architecture_name ]
   [ designID | -auto ]
   [ -config configID ]
   [ -parameter value ]
```

Data Types

```
architecture_name string
designID string
configID string
value string
```

ARGUMENTS

-vhdl_arch *architecture_name*

Specifies a VHDL target architecture for elaboration. The architecture name you specify must be a valid VHDL architecture.

designID

Specifies the non-VHDL design to link or elaborate.

-auto

Attempts to automatically detect the top-level design.

-config *configID*

Specifies a Verilog Configuration for elaboration. The configID is the name of a valid Verilog configuration. If user has not specified a designID, Formality will pick first design in the design statement of the configuration. Formality will not accept -auto with -config option.

-parameter *valueID*

Specifies a new value for design parameters. The parameter values may be integers or specified in the following format:

```
<param_name> <hex value format> <base>'h<value>
```

For example:

```
fm_shell (setup)> set_top ref:/WORK/design \
                  -param {w1 = 4'h5 }
```

For proper usage, use the following rules:

- `Single positional parameter: param_value`

-
  ```
  Multiple positional parameters:
       { param_value1, param_value2, ... }
  ```

-
  ```
  Single name-based parameters:
       { param_name = param_value }
       { param_name <= param_value }
  ```

-
  ```
  Multiple name-based parameters:
       { param_name1 = param_value1, param_name2
         param_value2, ... }
       { param_name1 <= param_value1, param_name2 <=
         param_value2, ... }
  ```

-
  ```
  Mixed positional and name-based parameters:
       { param_value1, param_value2, param_name3 =
         param_value3, param_name4 = param_value4, ... }
       { param_value1, param_value2, param_name3 <
         param_value3, param_name4 <= param_value4, ... }
  ```

You cannot specify a name-based parameter before a positional parameter. This is consistent with Design Compiler.

DESCRIPTION

This command resolves all name-based design references in a specific design.

When resolving references, Formality does not look outside the container in which the specified design resides. If you are linking a netlist, Formality attempts to resolve all named references. If you are linking an RTL or VHDL design, issuing the **set_top** command causes Formality to first elaborate the design and then attempt to resolve all named references.

If you are elaborating VHDL and you have more than one architecture, you can use the **-vhdl_arch** option to supply the target architecture.

If you do not specify the *designID* argument or specify the **-auto** option, Formality attempts to use the current design. If you specify the *designID* argument without the path prefix, the **set_top** command attempts to find it in the default library in the current container.

During the linking process, Formality issues errors if it cannot resolve a named reference. Alternatively, if you set the **hdlin_unresolved_modules** variable to **black_box**, Formality issues warnings and treats unresolved named references as black boxes.

You can also change the design parameter values by using the **-parameter** *value* option.

For the **-parameter** option, the parameter specification must be specially delimited. The parameter specification can be enclosed in double quotation marks ("), but this is not advised if the specification itself includes string values. Instead, use the special parameter-delimiter, which brackets the parameter specification syntax. The parameter-delimiter starts and ends with `#' (hash). For multi-line strings put backslashes (\) as the last character in the line. This is because "#" is properly recognized as a string constant.

Spaces can be inserted in the middle of the parameter string to make your script easier to read, but you must either enclose the parameter string in double quotation marks ("), or escape the spaces using back slashes (\).

NOTE: After successful completion of the **set_top** command all shared libraries in the container are no longer shared.

EXAMPLES

The following example loads a design and attempts to link it without the supporting library. Formality issues warning messages for each cell in the design whose name cannot be resolved. Formality then reloads with the required supporting library and issues the same **set_top** command.

```
fm_shell> read_db mapped_gate_lca500k.db
Loading db file 'mapped_gate_lca500k.db'
No target library specified, default is WORK
1
fm_shell> set_top ref:/WORK/CORE
Warning: Cannot link cell 'OUTPUT_BUFFER/U11' to its
reference design 'IV'.  (FE-LINK-2)
Warning: Cannot link cell 'MULTIPLEXOR/U15' to its
reference design 'IV'.  (FE-LINK-2)
Warning: Cannot link cell 'MULTIPLEXOR/U16' to its
reference design 'IV'.  (FE-LINK-2)
...
set_top of 'ref:/WORK/CORE' failed
```

```
fm_shell> read_db mapped_gate_lca500k.db
Loading db file 'mapped_gate_lca500k.db'
No target library specified, default is WORK
1
fm_shell> read_db lca500k.db
Loading db file 'lca500k.db'
1
fm_shell> link ref:/WORK/CORE
Set top 'ref:/WORK/CORE' completed successfully
1
```

The following example elaborates the design using a configuration specified with set_top command

```
fm_shell> set_top r:/rtlLib3/top -config  r:/cfgLib/cfg_top
Setting top design to 'r:/RTLLIB3/top'
...
```

If user hs not specified the designID, Formality will choose first design in the 'design statement' of configuration to elaborate.

```
fm_shell> set_top -config  r:/cfgLib/cfg_top
Setting top design to 'r:/RTLLIB/mid'
...
```

SEE ALSO

```
verify(2)
hdlin_unresolved_modules(3)
```

set_user_match

Creates pairs of matched points when two or more comparable design objects are specified.

SYNTAX

```
status set_user_match
   [ -type ID_type_list ]
   [ -inverted | -noninverted ]
   objectID_1 objectID_2
```

Data Types

```
ID_type_list string
objectID string
```

ARGUMENTS

-type ID_type_list

Specifies the object type. You may use this option if the name of a specified design object is associated with more than one object type in the same design. Specify one of the following values for the *ID_type* argument:

- *pin* - Specifies pin type
- *port* - Specifies port type
- *net* - Specifies net type
- *cell* - Specifies cell type
- *{pin net}* - Specifies that the first object is a pin and the second is a net
- *{net pin}* - Specifies that the first object is a net and the second is a pin

objectID_1 objectID_2

Matches the design objects specified. The design objects must be comparable in type. Exactly one of the objects must be from the reference. The others must be from the implementation. If you specify a name consisting of a regular expression that resolves to more than one object, the tool reports an error. However, if the name resolves to multiple objects with identical names (and you do not specify the object type), only one of these objects will be affected. The precedence in this case is pin, port, net, cell.

-inverted

Specifies that the matched design objects have inverted relationship. Default is unknown relationship.

-noninverted

Specifies that the matched design objects have noninverted relationship. Default is unknown relationship.

DESCRIPTION

This command matches two or more design objects. You can use this command to match points that Formality could not match during its matching process. For example, Formality does not automatically match primary output ports that differ in name but are otherwise comparable.

Specify the polarity of the match using the **-inverted** or the **-noninverted** options. If the **verification_inversion_push** variable is enabled, Formality determines the polarity for registers that have unspecified polarities. If the **verification_inversion_push** is not enabled, then all unknown polarities will default to noninverted. For I/O ports any unknown polarity is assumed noninverted.

User-defined matches are not applied to the designs until the next *match* or *verify* command. Many other related points may be matched as a result of a user-defined match. Once a user-defined match is applied, it cannot be removed unless you unapply it using the *undo_match* or *setup* command. This prevents situations where the matched and setup states are inconsistent. You can determine the matches resulting from a *set_user_match* command by following it with the *match* and *report_matched_points -last* commands.

One reference design object may be matched with more than one implementation design object. In that case, each implementation object is verified against the reference object. A single implementation object may not match more than one reference object. One-to-many matches are not discovered automatically by Formality, but may only be specified by the user with **set_user_match**. Hierarchical blocks (linked cells) and their pins may not be multiply matched, except for black boxes and black box pins.

EXAMPLES

This example creates a compare point using two ports: *FULL* and *A_FULL*. The design containing the port *FULL* is the current design. The port *A_FULL* is located in the impl container, the WORK design library, and the CORE design.

```
fm_shell> set_user_match impl:/WORK/CORE/A_FULL \
FULL
```

This example matches registers with the same name as the ports in the previous example. Consequently, the **-type** option is required.

```
fm_shell> set_user_match -type cell \
impl:/WORK/CORE/A_FULL \
FULL
```

SEE ALSO

```
match(2)
undo_match(2)
remove_user_match(2)
report_matched_points(2)
report_unmatched_points(2)
```

set_verify_points

Marks multiple compare points for verification.

SYNTAX

```
status set_verify_points
   [ -type ID_type ]
   [ -file fileOfObjectIDs ]
   [ objectID_list ]
```

Data Types

```
ID_type string
fileOfObjectIDs string
objectID_list string
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

-type *ID_type*

Specifies the type for the specified objects. Use this option if the name of the specified design object is associated with more than one object type in the same design. Specify one of the following object types:

- *pin* - To specify pin type
- *port* - To specify port type
- *net* - To specify net type
- *cell* - To specify cell type

`-file` ***`fileOfObjectIDs`***

Specifies a file containing object IDs. See the *objectID_list* argument description for details on object IDs.

`objectID_list`

Specifies one or more design objects associated with the compare point to verify. If you specify a name consisting of a wildcard expression that resolves to more than one object, the operation is applied to all of the matching objects. However, if the name resolves to multiple objects with identical names and you do not specify the object type, only one of these objects is affected in the following precedence: pin, port, net, cell.

DESCRIPTION

This command marks multiple compare points for verification.

Verify points affect the subsequent **match** and **verify** commands. To view the curent set of compare points marked for verification, use the **report_verify_points** command.

EXAMPLES

The following example verifies only the specified ports in the reference design named top.

```
fm_shell (setup)> set_verify_points {r:/WORK/top/out1 r:/WORK/top/out2 r:/WORK/top/out3}
```

The following example uses a wildcard character to enable verification of all ports matching the expression:

```
fm_shell (setup)> set_verify_points -type port {r:/WORK/top/out*}
```

The following example writes a file of failing points and then later enables verification of only those points:

```
fm_shell (verify)> report_failing_points -list > fail.fpt
[...later in a separate Formality run...]
fm_shell (setup)> set_verify_points -file fail.fpt
```

SEE ALSO

```
remove_verify_points(2)
report_verify_points(2)
report_aborted_points(2)
report_failing_points(2)
report_unmatched_points(2)
```

set_vsdc

Sets the VSDC.

SYNTAX

```
status set_vsdc
[-append]
[-ordered]
[-extension name]
[filedirnames]
```

Data Types

```
name string
filedirnames string
```

Enabled Shell Modes

Setup
Guide

ARGUMENTS

-append

Appends the data to the current VSDC information.

-ordered

Preserves the ordering of the filenames when reading.

-extension *name*

Specifies the extension to be used when recursively searching directories for VSDC files. Default is "vsdc".

filedirnames

Specifies the name of the VSDC files to read or directories to search. If nothing is specified, the VSDC is reset.

DESCRIPTION

This command sets the VSDC. The VSDC provides valuable information that is used during compare point matching to facilitate alignment of compare points in the designs to be verified. Unlike SVF, which contains information that is either dependent on Synopsys-specific technology or is only in limited customer availability, VSDC files are intended for all verification flows.

If multiple files are specified, Formality attempts to order the files before processing (unless -ordered is specified in which case Formality does not change the order). If a directory name is specified, Formality recursively searches the directory and reads any file whose extension matches that specified using the **-extension** option or "vsdc" if no extension is specified. For each VSDC file found, the contents are processed and all information is stored in memory for later use during name-based compare point matching.

If a file name is not specified, the VSDC is reset. However, it is preferred to use the **remove_guidance** command to delete the stored VSDC data.

You can see the contents of a previously set VSDC by using the **report_guidance -to** *filename* command.

EXAMPLES

The following example shows how to use the **set_vsdc** command.

```
fm_shell (setup)> set_vsdc myfile.vsdc
VSDC set to 'myfile.vsdc'.
1
```

SEE ALSO

```
set_svf(2)
report_guidance(2)
remove_guidance(2)
```

setenv

Sets the value of a system environment variable.

SYNTAX

```
string setenv variable_name new_value

string variable_name


string new_value
```

ARGUMENTS

variable_name

Names of the system environment variable to set.

new_value

Specifies the new value for the system environment variable.

DESCRIPTION

The **setenv** command sets the specified system environment *variable_name* to the *new_value* within the application. If the variable is not defined in the environment, the environment variable is created. The **setenv** command returns the new value of *variable_name*. To develop scripts that interact with the invoking shell, use **getenv** and **setenv**.

Environment variables are stored in the Tcl array variable **env**. The environment commands **getenv**, **setenv**, and **printenv** are convenience functions to interact with this array.

The **setenv** command sets the value of a variable only within the process of your current application. Child processes initiated from the application using the **exec** command after a usage of **setenv** inherit the new variable value. However, these new values are not exported to the parent process. Further, if you set an environment variable using the appropriate system command in a shell you invoke using the **exec** command, that value is not reflected in the current application.

EXAMPLES

The following example changes the default printer.

```
shell> getenv PRINTER
laser1
shell> setenv PRINTER "laser3"
laser3
shell> getenv PRINTER
laser3
```

SEE ALSO

exec(2)
getenv(2)
unsettenv(2)
printenv(2)
printvar(2)
set(2)
sh(2)
unset(2)

setup

Causes Formality to revert to SETUP mode.

SYNTAX

```
status setup
```

Enabled Shell Modes

Match
Verify

DESCRIPTION

This command changes the Formality mode to the Setup mode, discards any matching and verification results, and enables all setup commands.

Upon invocation, Formality starts in the Setup mode. After executing the **match** command, Formality switches to the Match mode. After executing the **verify** command, Formality switches to the Verify mode.

Many commands can be executed only in the Setup mode. Execute the **setup** command to return to the Setup mode, which re-enables all setup commands and discards all matching information resulting from the **match** and **verify** commands. . The prompt changes to reflect the current mode, as follows:

```
fm_shell (guide)>
fm_shell (setup)>
fm_shell (match)>
fm_shell (verify)>
```

EXAMPLES

The following example illustrates returning to the Setup mode from the Match mode.

```
fm_shell (match)> setup
```

```
      1
      fm_shell (setup)>
```

SEE ALSO

```
guide(2)
match(2)
verify(2)
```

sh

Executes a command in a child process.

SYNTAX

```
sh
[args]
```

Data Types

```
args string
```

ARGUMENTS

args

Command and arguments that you want to execute in the child process.

DESCRIPTION

Use this command to execute a command in a child process. This is very similar to the UNIX **exec** command. However, file name expansion is performed on the arguments.

EXAMPLES

This example shows how you can remove files using a wildcard.

```
fm_shell> ls aaa*
aaa1      aaa2      aaa3
fm_shell> sh rm aaa*
fm_shell> ls aaa*
Error: aaa*: No such file or directory Use error_info for more info. (CMD-013)
```

SEE ALSO

sizeof_collection

Returns the number of objects in a collection.

SYNTAX

```
int sizeof_collection
   collection1
```

Data Types

collection1	collection

ARGUMENTS

collection1

Specifies the collection for which to get the number of objects. If the empty collection (empty string) is used for the *collection1* argument, the command returns 0.

DESCRIPTION

The **sizeof_collection** command is an efficient mechanism for determining the number of objects in a collection.

EXAMPLES

The following example from Formality shows a simple way to find out how many objects matched a particular pattern and filter in the **get_cells** command.

```
prompt> echo "Number of techlib cells: \
?              [sizeof_collection \
?                [get_cells -hierarchical -filter is_techlib]]"
Number of techlib cells: 10
```

SEE ALSO

```
collections(2)
get_cells(2)
```

sort_collection

Sorts a collection based on one or more attributes, resulting in a new, sorted collection. The sort is ascending by default.

SYNTAX

```
collection sort_collection
   [-descending]
   [-dictionary]
   collection
   criteria
```

Data Types

```
   collection           collection
   criteria             list
```

ARGUMENTS

-descending

Indicates that the collection is to be sorted in reverse order. By default, the sort proceeds in ascending order.

-dictionary

Sort strings dictionary order. For example "a30" would come after "a4".

collection

Specifies the collection to be sorted.

criteria

Specifies a list of one or more application or user-defined attributes to use as sort keys.

DESCRIPTION

You can use the **sort_collection** command to order the objects in a collection based on one or more attributes. For example, to get a collection of non-techlib cells increasing alphabetically, followed by techlib cells increasing alphabetically, sort the collection of cells using the *is_techlib* and *full_name* attributes as *criteria*.

In an ascending sort, Boolean attributes are sorted with those objects first that have the attribute set to *false*, followed by the objects that have the attribute set to *true*. In the case of a sparse attribute, objects that have the attribute come first, followed by the objects that do not have the attribute.

Sorts are ascending by default. The *-descending* option reverses the order of the objects.

EXAMPLES

The following example from Formality sorts a collection of cells based on the is_techlib attribute, and adds a second key to list them alphabetically. In this example, cells U1, U2 and U10 are techlib cells, and Z1 and Z2 are non-techlib cells. Because the *is_techlib* attribute is Boolean, those objects with the attribute set to *false* are listed first in the sorted collection.

```
pt_shell> set zc [get_cells {Z2 U2 Z1 U1 U10}]
{r:/WORK/top/Z1 r:/WORK/top/U2 r:/WORK/top/Z1 r:/WORK/top/U1 r:/WORK/top/U10}
pt_shell> set zsort [sort_collection $zc {is_techlib full_name}]
{r:/WORK/top/Z1 r:/WORK/top/Z2 r:/WORK/top/U1 r:/WORK/top/U10 r:/WORK/top/U2}
pt_shell> set zsort [sort_collection -dictionary $zc {is_techlib full_name}]
{r:/WORK/top/Z1 r:/WORK/top/Z2 r:/WORK/top/U1 r:/WORK/top/U2 r:/WORK/top/U10}
```

SEE ALSO

```
collections(2)
```

source

Reads a file and evaluates it as a Tcl script.

SYNTAX

```
source
   [-echo]
   [-verbose]
   file
```

Data Types

```
file string
```

ARGUMENTS

-echo

Echoes each command as it is executed. Note that this option is a non-standard extension to Tcl.

-verbose

Displays the result of each command executed. Note that error messages are always displayed. This option is a non-standard extension to Tcl.

file

Specifies the script file to read.

DESCRIPTION

This command reads a command from a file and then pass each one to the command interpreter. The result of the **source** command is the result of the last command executed from the file specified.

- If an error occurs in evaluating the contents of the file, the **source** command displays that error.
- If the **return** command is invoked from within the file, the remainder of that file is skipped and the **source** command returns normally with the result from the **return** command.

By default, the **source** command displays little information. It is possible to get intermediate information from the **source** command using the **-echo** and **-verbose** options. The **-echo** option echoes each command as it appears in the script. The **-verbose** option echoes the result of each command after execution. To emulate the behavior of the Design Compiler **include** command, use both these options.

The file name specified can be a fully expanded file name and can begin with a tilde. Under normal circumstances, this file is searched for based only on what you typed. However, if the system variable **sh_source_uses_search_path** is set to "true", the file is searched for based on the path established with the **search_path** variable.

EXAMPLES

This example reads in a script of aliases.

```
fm_shell> source -echo aliases.tcl
alias q quit
alias hv {help -verbose}
alias include {source -echo -verbose}
```

SEE ALSO

```
search_path(3)
sh_source_uses_search_path(3)
```

start_gui

Starts a Formality GUI session.

SYNTAX

```
status start_gui
```

DESCRIPTION

This command starts a GUI session.

Once a GUI window has been created, subsequent **start_gui** commands are ignored until after using the **stop_gui** command. The GUI can be started and stopped any number of times during a single Formality session.

SEE ALSO

```
stop_gui(2)
```

stop_gui

Closes the Formality GUI session.

SYNTAX

```
status stop_gui
```

DESCRIPTION

This command closes the current Formality GUI session.

If the GUI window is not currently open, the **stop_gui** command is ignored. The GUI can be started and stopped any number of times during a single Formality session.

SEE ALSO

```
start_gui(2)
```

suppress_message

Disables printing of one or more informational or warning messages.

SYNTAX

```
string suppress_message [message_list]

list message_list
```

ARGUMENTS

message_list

A list of messages to suppress.

DESCRIPTION

The **suppress_message** command provides a mechanism to disable the printing of messages. You can suppress only informational and warning messages. The result of **suppress_message** is always the empty string.

A given message can be suppressed more than once. So, a message must be unsuppressed (using **unsuppress_message**) as many times as it was suppressed in order for it to be enabled. The **print_suppressed_messages** command displays the currently suppressed messages.

EXAMPLES

When the argument to the **unalias** command does not match any existing aliases, the CMD-029 warning message displays. This example shows how to suppress the CMD-029 message:

```
prompt> unalias q*
Warning: no aliases matched 'q*' (CMD-029)
prompt> suppress_message CMD-029
prompt> unalias q*
prompt>
```

SEE ALSO

print_suppressed_messages(2)
unsuppress_message(2)
get_message_ids(2)
set_message_info(2)

test_compare_rule

Tests a name matching rule on current unmatched points or user-specified names.

SYNTAX

```
status test_compare_rule
   [designID | -r | -i]
   -name name_list
   -from search_pattern
   -to replace_pattern
   [-substring string]
   [-type type]
```

Data Types

```
   search_pattern string
   replace_pattern string
   string string
   type string
   name_list string
```

ARGUMENTS

designID

Specifies the design on which to test unmatched compare points by the name matching rule.

-r

Specifies the reference design for testing.

-i

Specifies the implementation design for testing.

-name

Specifies a list of arbitrary user-defined names for testing. No designID should be specified along with this option.

-to *replace_pattern*

Specifies the pattern Formality uses as a replacement after successfully finding the *search_pattern* argument. Formality supports the same syntax for *replace_pattern* as it does for regular expressions used by the UNIX **sed** command. Arithmetic expressions can be specified delimited by \(and \).

-from *search_pattern*

Specifies the pattern for which Formality searches during the compare point creation process. Formality supports the same syntax for *search_pattern* as it does for regular expressions used by the UNIX **sed** command. Please make sure that only relative names are used in the pattern, excluding top level name.

-substring *string*

Reports points containing the specified *string*.

-type *ID_type*

Specifies the object type to be tested by the rule:

- cell
- port
- net

DESCRIPTION

This command tests name translation rules. Note that this command just tests the rule. If the testing is successful, use the set_compare_rule command to set the compare rule. You can perform testing on current unmatched points or user specified arbitrary names. A report is printed with mapped and unmapped names.

Compare rules are based on a regular expression syntax identical to that supported by the UNIX **sed** facility. This command also supports arithmetic expressions in the replace pattern. You can use this command to check the syntactic correctness of your regular expressions and arithmetic expressions.

You can specify a particular design to be tested by supplying a *designID*. If you do not supply a *designID*, Formality tests the rule(s) on the current design. Also, you can omit the container portion of the *designID* to default to the current container.

EXAMPLES

This example tests a compare rule for the design 'topxor'. The rule performs the following name transformations: out0 -> out_1_reg out1 -> out_2_reg out2 -> out_3_reg out3 -> out_4_reg

.nf

fm_shell> **test_compare_rule r:/WORK/topxor** \ **-from {out\([0-9]+\)} -to {out_\(\1+1\)_reg}**

3 Mapped Point(s):

Port r:/WORK/topxor/out1 mapped to r:/WORK/topxor/out_2_reg

Port r:/WORK/topxor/out4 mapped to r:/WORK/topxor/out_5_reg

Port r:/WORK/topxor/out9 mapped to r:/WORK/topxor/out_10_reg

2 Unmapped Point(s):

Pin r:/WORK/topxor/clock

Net r:/WORK/topxor/wire0

Total: 5 (3 Mapped, 1 Unmapped) 1

This example tests the previous rule on arbitrary names.

```
fm_shell> test_compare_rule -name {out1 reg clk vcc out4}    \
          -from {out\([0-9]+\)} -to {out_\(\1+1\)_reg}

2 Mapped Point(s):

    Name          r:/WORK/topxor/out1
     mapped to    r:/WORK/topxor/out_2_reg

    Name          r:/WORK/topxor/out4
     mapped to    r:/WORK/topxor/out_5_reg

2 Unmapped Point(s):

    Name          r:/WORK/topxor/reg

    Name          r:/WORK/topxor/clk

Total: 4 (2 Mapped, 2 Unmapped)
1
```

SEE ALSO

```
set_compare_rules(2)
report_unmatched_points(2)
```

translate_instance_pathname

Translates an instance-based pathname to a Formality design ID or object ID.

SYNTAX

```
string translate_instance_pathname
   [ -type ID_type ] pathname
```

Data Types

```
ID_type string
pathname string
```

ARGUMENTS

-type *ID_type*

Specifies the object type for *pathname*'s leaf. Use this switch when the leaf portion of the instance-based pathname could resolve to more than one type of design object. Specify one of the following values for *ID_type*:

- *cell*
- *net*
- *port*
- *pin*

pathname

Specifies the instance-based pathname of the design object you want to translate. Use the following syntax when supplying a value for *pathname*:

```
[[[container:]/des_lib/design/]inst_1[/ inst_n...]/]cell_name
[[[container:]/des_lib/design/]inst_1[/ inst_n...]/]net_name
[[[container:]/des_lib/design/]inst_1[/ inst_n...]/]port_name
```

```
[[[container:]/des_lib/design/]inst_1[/ inst_n...]/]pin_name
```

DESCRIPTION

This command translates an instance-based design object pathname into the equivalent Formality *designID* or *objectID*.

You can create objectIDs for cells, nets, ports, and pins. When specifying the instance-based pathname, Formality uses the current container if you do not include the container in the instance *pathname* but do supply design library and design names. If you omit the container, design library, and design, Formality uses the current design.

When translating an object that shares its name with another object of a different type, you must use the -type option to supply the object type. Otherwise, Formality resolves the conflict using the following order: cell, port, net, then pin.

EXAMPLES

The following example returns the objectID for the port CC in the current design.

```
fm_shell> translate_instance_pathname -type port CC
ref:/WORK/CORE/CC
```

SEE ALSO

```
find_cells(2)
find_nets(2)
find_pins(2)
find_ports(2)
```

unalias

Removes one or more aliases.

SYNTAX

```
unalias
   pattern
```

Data Types

```
pattern string
```

ARGUMENTS

pattern

Removes aliases matching the specified pattern. This argument can be repeated. Each is the name of a specific alias to remove, or a pattern containing the wildcard characters "*" and "%" that match one or more aliases to be removed.

DESCRIPTION

This command removes aliases created using the **alias** command.

EXAMPLES

This command removes all aliases.

```
fm_shell> unalias *
```

This command removes all aliases beginning with f, and the alias *rt100*.

```
fm_shell> unalias f* rt100
```

SEE ALSO

```
alias(2)
```

undo_edits

Reverts the changes made by edit commands.

SYNTAX

```
status undo_edits
```

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command reverts the changes made to the design using edit commands such as the **create_net** command.

The design is reverted to its state after the last **commit_edits** command. If no **commit_edits** commands have been issued, the circuit is reverted back to its state before the edit commands were issued.

Outside of Setup mode, this command only works on edit designs created using the **edit_design** command.

EXAMPLES

The following example creates a new net in design r:/WORK/mid and reverts the change again.

```
fm_shell (setup)> current_design r:/WORK/mid
fm_shell (setup)> create_net N42
fm_shell (setup)> undo_edits
```

SEE ALSO

```
commit_edits(2)
create_nets(2)
current_design(2)
edit_design(2)
```

undo_match

Removes matches between design objects that are matched.

SYNTAX

```
status undo_match
   [ -all ]
```

Enabled Shell Modes

Match
Verify

ARGUMENTS

-all

Removes matches from all previously matched compare points.

DESCRIPTION

This command removes matches made by the **match** command. Run the **undo_match** command again to unmatch the next previous batch of matches. Use the **undo_match -all** command to remove all matches by all previously issued the **match** commands.

After running the **undo_match** command, Formality is still in the Match mode. To return to the Setup mode, run the **setup** command.

If you used the **set_user_match** command to establish matching directives, you must use the **remove_user_match** command to remove them. Otherwise, the next **match** or **verify** command will cause th objects to be matched again.

The **undo_match** command is intended to help you debug matching problems. For example, you can set a compare rule, run **match**, view the results with **report_matched_points** and **report_unmatched_points** command, and if the compare rule caused some points to be incorrectly matched, run the **undo_match** command and fix the compare rule.

SEE ALSO

```
match(2)
setup(2)
report_matched_points(2)
report_unmatched_points(2)
set_user_match(2)
remove_user_match(2)
report_user_matches(2)
verify(2)
```

ungroup

Removes a level of hierarchy.

SYNTAX

```
ungroup
   cell_list | -all
   [-prefix prefix_name]
   [-flatten] [-simple_names]
```

Data Types

```
cell_list string
prefix_name string
```

Enabled Shell Modes

Setup

ARGUMENTS

cell_list

Specifies a list of cells in the current design that are to be ungrouped. The contents of these cells are brought up to the same level as the cells. You must specify either the *cell_list* or the **-all** option, but not both.

-all

Indicates that all cells in the current_design are to be ungrouped. You must specify either *cell_list* or -all, but not both.

-prefix *prefix_name*

Specifies the prefix to use in naming ungrouped cells. The default naming style is:

`cell_being_ungrouped/`*`old_cell_name`*`{number}`

To provide a vestige of the former hierarchy, omit this option when the **-flatten** option is used.

-flatten

Removes the hierarchy of the specified cell and its subcells recursively until all levels of hierarchy are removed.

-simple_names

Uses simple, non-hierarchical names for cells that are ungrouped. Unless this option is used, cells are given default hierarchical names. With this option, cells maintain their original names.

DESCRIPTION

Removes a single level of hierarchy from the current design by exploding the contents of the specified cell in the current design.

New cell names are created by concatenating *prefix_name* with original cell names. If the resulting name is not unique, the new unique name is:

`prefix_name``cell`*`number`*

If you do not specify a prefix, the default is:

`cell_being_ungrouped/`*`old_cell_name`*`{`*`number`*`}`

EXAMPLES

This example ungroups a specified list of cells.

```
fm_shell (setup)> ungroup {u1 u2 u3}
```

This example ungroups a specific cell and specifies the prefix to be used.

```
fm_shell (setup)> ungroup {u1} -prefix "U1:"
```

This example completely removes the hierarchy of two cells.

```
fm_shell (setup)> ungroup {u1 u2} -flatten
```

This example completely collapses the hierarchy of the current_design.

```
fm_shell (setup)> ungroup -all -flatten
```

SEE ALSO

```
current_design(2)
```

uniquify

Creates unique design names for multiply-instantiated designs in hierarchical designs.

SYNTAX

```
status **uniquify**
   [ *designID* ]
```

Data Types

```
designID string
```

Enabled Shell Modes

Setup

ARGUMENTS

designID_1

The design for which you want to create unique design name.

DESCRIPTION

This command removes multiply-instantiated hierarchy in a design by creating a unique design for each cell instance. When you enter the command, Formality searches for designs referenced in the hierarchy of the design and generates new, uniquely named designs for all instances that have the same name.

Having unique design names is necessary any time you use a Formality command to specify a particular instance of a multiply instantiated design. For example, when creating a compare point, if the design object you want to use from the implementation design is instantiated several times in the top-level design. Specifying the design name does not differentiate between instances. Before you can create the compare point, you need to create unique design names in the implementation design. If you do not specify the *designID* argument with the command, Formality uses the current design.

EXAMPLES

This example creates unique design names in the top-level design named *impl:/WORK/CORE*.

```
fm_shell (setup)> uniquify
Uniquifying design 'impl:/WORK/CORE'
Uniquify of 'impl:/WORK/CORE' completed successfully
1
```

SEE ALSO

unread_analysis

The **unread_analysis** command is used to determine whether any failing compare point is functionally unread. The command updates the verification status of any unread point, and returns 1 (success) if all analyzed failing points are unread

SYNTAX

```
unread_analysis
   [ -list ref_compare_point_list ]
   [ -level integer ]
```

ARGUMENTS

-list *ref_compare_point_list*

Runs unread analysis on a list of failing reference compare points. If this option is not specified, unread analysis is run on all failing compare points.

-level *integer*

Specifies how deep to look for observability. If a compare point is read by a downstream point, then that point needs to be analyzed to determine whether it is read by its downstream readers, and so forth. This variable determines the depth of this traversal before declaring a point to be READ.

DESCRIPTION

unread_analysis determines whether any given failing compare point(s) are unobservable. It does so by unmatching the compare point and verifying whether any downstream points subsequently fail verification. If no downstream points are affected by the unmatched point (i.e., successful verification while the compare point is unmatched), the compare point is declared UNREAD, and marked as such in the database.

If a downstream primary output port or black box input fails verification from an unmatched compare point, then the compare point is considered observable, and marked as READ.

If any other downstream compare points fail verification, they must subsequently be analyzed to determine if they are READ by their fanout. This traversal continues through *-levels* of compare points. If any remaining compare points still fail verification after the traversal has reached this limit, the original point is considered READ.

If any downstream compare point is encountered that was an failing point in the original verification, then its verification will continue to fail after the original compare point is unmatched. For this reason, the original point will be declared UNREAD.

unread_analysis will work on all failing points, unless a subset of these points is provided through the *-list* option.

If all analyzed points are determined to be UNREAD, the command returns 1 (success). If any points are READ, then they are legitimate failing points, and the command returns 1.

EXAMPLE

This is a sample output showing the analysis of an unread and a read compare point, with the final summary:

```
fm_shell> unread_analysis

Analyzing unread compare point: r:/WORK/mychip/myblock/abc/def/keyval_reg
Level 0: UNREAD

Analyzing unread compare point: r:/WORK/mychip/instr_reg
Level 0: 16 potential readers
Level 1: READ, 66 readers

Unread analysis cleanup

New unread points:
  r:/WORK/mychip/myblock/abc/def/keyval_reg

fm_shell>
```

SEE ALSO

```
verify
```

unsetenv

Removes a system environment variable.

SYNTAX

```
string getenv
   variable_name
```

Data Types

```
variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of the environment variable to be unset.

DESCRIPTION

The **unsetenv** command searches the system environment for the specified *variable_name* and removes variable from the environment. If the variable is not defined in the environment, the command returns a Tcl error. The command is catchable.

Environment variables are stored in the **env** Tcl array variable. The **unsetenv**, commands is a convenience function to interact with this array. It is equivalent to 'unset ::env(*variable_name*)'

The application you are running inherited the initial values for environment variables from its parent process (that is, the shell from which you invoked the application). If you unset the variable using the **unsetenv** command, you remove the variable value in the application and in any new child processes you initiate from the application using the **exec** command. However, the variable is still set in the parent process.

See the **set** and **unset** commands for information about working with non-environment variables.

EXAMPLES

In the following example, **unsetenv** remove the DISPLAY varible from the environment:

```
prompt> getenv DISPLAY
host:0
prompt> unsetenv DISPLAY
prompt> getenv DISPLAY
Error: can't read "::env(DISPLAY)": no such variable
        Use error_info for more info. (CMD-013)
```

SEE ALSO

```
catch(2)
exec(2)
printenv(2)
set(2)
unset(2)
setenv(2)
getenv(2)
```

unsuppress_message

Enables printing of one or more suppressed informational or suppressed warning messages.

SYNTAX

```
string unsuppress_message [messages]

list messages
```

ARGUMENTS

messages

A list of messages to enable.

DESCRIPTION

The **unsuppress_message** command provides a mechanism to re-enable the printing of messages which have been suppressed using **suppress_message**. You can suppress only informational and warning messages, so the **unsuppress_message** command is only useful for informational and warning messages. The result of **unsuppress_message** is always the empty string.

You can suppress a given message more than once. So, you must unsuppress a message as many times as it was suppressed in order to enable it. The **print_suppressed_messages** command displays currently suppressed messages.

EXAMPLES

When the argument to the **unalias** command does not match any existing aliases, the CMD-029 warning message displays. This example shows how to re-enable the suppressed CMD-029 message. Assume that there are no aliases beginning with 'q'.

```
prompt> unalias q*
prompt> unsuppress_message CMD-029
prompt> unalias q*
Warning: no aliases matched 'q*' (CMD-029)
```

SEE ALSO

print_suppressed_messages(2)
suppress_message(2)

verify

Verifies the functional equivalence between the two specified designs or two comparable design objects.

SYNTAX

```
status verify
   [ designID_1 designID_2 ] |
   [[ -inverted ]
    [ -type ID_type ]
     objectID_1 objectID_2 ] |
    [[ -constant0 | -constant1 ] [ -type ID_type ] objectID ] |
    [[ -type ID_type ] objectID [ -constant0 | -constant1 ]] |
   [ -probe ] |
   [ -restart | -incremental ]
   [-level integer]
```

Data Types

```
ID_type string
objectID string
ID_type string
integer integer
```

ARGUMENTS

designID_1

Specifies the reference design. If you do not specify a design, Formality uses the current reference design.

designID_2

Specifies the implementation design. If you do not specify a design, Formality uses the current implementation design.

-inverted

Verifies the inverse-equivalence of the two specified objects.

`-type` ***`ID_type`***

Specifies an object type. You must use this option if the leaf portions of the instance-based pathnames of *objectID_1* and *objectID_2* resolve to more than one type of design object. Specify one of the following values for the *ID_type* argument:

- **`cell`**
- **`net`**
- **`port`**
- **`pin`**

`objectID_1`

Specifies a design object that is comparable to *objectID_2* and is of the same type as *objectID_2*.

`objectID_2`

Specifies a design object that is comparable to *objectID_1* and is of the same type as *objectID_1*.

`-constant0`

Specifies a constant 0. This is used to check if the objectIDs are constant 0. If this switch is specified on the command line before the object ID, then the constant is considered the reference. If this switch is specified on the command line after the Object ID, then the constant is considered the implementation.

`-constant1`

Specifies a constant 1. This is used to check if the objectIDs are constant 1. If this switch is specified on the command line before the Object ID, then the constant is considered the reference. If this switch is specified on the command line after the Object ID, then the constant is considered the implementation.

`-probe`

Verifies only probes points. This is used to verify probe net pairs set using the **set_probe_points** command. This debug option is allowed only when main verification is partially or fully done and shell is in the Verify mode. Probe verification do not alter or destroy the current match or verify results.

`-restart`

Discards any verification results from previous verify commands; verify all compare points. Opposite of **-incremental**. The default behavior is **-incremental**, which indicates to Formality so save partial results from successive verifications.

-incremental

Opposite of **-restart**. The default behavior is **-incremental**, which indicates to preserve any results from previous verify commands and to verify only unverified points. If matching changes have been made since the last verify command, affected compare points will be moved to the unverified state before being verified. If verification effort level has been increased also verify compare points aborted due to complexity.

-level *integer*

Assumes functional boundaries at blocks at or above this level only have been preserved, where top is level 0. The specified integer must be greater than or equal to zero. The default is no limit. If you know that the boundaries of blocks below a certain level have different functions in the reference and implementation designs, you can use this switch to improve verification performance. For example, if boundary optimization has occurred.

DESCRIPTION

This command proves or disproves design equivalence using the current implementation and reference designs, the specified implementation and reference designs, a specific compare point or user defined probe pairs.

In library_verification mode this command invokes verification on all the library cells that were selected for verification.

When you specify this command, Formality performs verification and reports its success. If you interrupt verification (Ctrl+C), Formality retains partial verification results. You can report these partial verification results.

Formality bases verification on the verification mode and applies all currently set environment parameters as well as any currently set design-specific parameters.

When verifying designs, Formality determines whether the designs are comparable and attempts to prove their functional equivalence. If all compare points prove equal when verifying two designs, Formality reports the designs as functionally equivalent (verification passes). However, should any compare point fail verification or abort during verification, Formality reports the two designs as functionally unequivalent (verification fails).

You can specify the implementation and reference designs, or let Formality use the current implementation and reference designs. Not supplying any design IDs causes Formality to use the current implementation and reference design for verification. Supplying two designIDs causes Formality to establish the first design as the current reference design and the second design as the current implementation design.

You must have successfully run the **set_top** command on the reference and implementation designs (or the top-level designs within which they are contained) before performing verification.

When verifying compare points, Formality determines whether the design objects constituting the compare points are comparable. Verification success or failure hinges on that compare point only.

If you are verifying a single compare point, you must supply two objectIDs that are of the same type (cell, port, or net). If more than one objectID exists in either design that shares the same name but is of a different type, you must use the **-type** option to differentiate the objects. The objects can be in different designs.

Pin-to-pin single compare point verification is only allowed on black box pins.

You can use the -probe option to verify debug probe pairs set using the **set_probe_points** command.

If verification fails, you can view a list of failing compare points by using the report_failing_points command. You can then direct Formality to use this list to diagnose the differences between the implementation and reference designs.

SEE ALSO

```
report_unverified_points(2)
report_aborted_points(2)
report_failing_points(2)
report_passing_points(2)
set_probe_points(2)
report_probe_status(2)
write_hierarchical_verification_script(2)
```

verify_edits

Copies the edited designs back to the original designs and performs a verification.

SYNTAX

```
status verify_edits
   [ -include_failures ]
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

DESCRIPTION

This command copies all the edited designs created by the **edit_design** command back to the original designs, and then verifies all compare points affected by the edits.

If the **-include_failures** option is used, the verification will also include all failing points. This is useful if your edits do not address all failures. This way those points that are not addressed by the edits will continue to fail.

This command changes the set of points targeted for verification. You can use **report_verify_points** to inspect the set, or **remove_verify_points** to prune it in preparation for a new verification.

Use the command **apply_edits** to copy the edited designs back to the original designs without doing a verification. This can only be done in setup mode, but **verify_edits** can be done in any mode.

EXAMPLES

Edit a design and verify the changes while in verify mode:

```
fm_shell (verify)> edit_design i:/WORK/bot
fm_shell (verify)> create_net ECO_NET_1
fm_shell (verify)> verify_edits
```

SEE ALSO

```
apply_edits(2)
create_net(2)
edit_designs(2)
remove_verify_points(2)
report_verify_points(2)
```

which

Locates a file and displays its pathname.

SYNTAX

```
which
   filename_list
```

Data Types

```
filename_list string
```

ARGUMENTS

filename_list

Specifies a list of files to locate.

DESCRIPTION

This command displays the location of the specified files. This command uses the **search_path** variable to find the location of the files. This command can be a useful prelude to the **read_db** or **link_design** commands, because it shows how these commands expand filenames. The **which** command can be used to verify that a file exists in the system.

If an absolute pathname is given, the command searches for the file in the given path and returns the full pathname of the file.

EXAMPLES

These examples are based on the following search_path.

```
fm_shell> set search_path "/u/foo /u/foo/test"
```

This command searches for the file name foo1 in the search_path.

```
fm_shell> which mydesign
/u/mypath/mydesign
```

SEE ALSO

```
search_path(3)
```

write_app_var

Writes a script to set the current variable values.

SYNTAX

```
string write_app_var
   -output file
   [-all | -only_changed_vars]
   [pattern]
```

Data Types

```
   file        string
   pattern     string
```

ARGUMENTS

-output *file*

Specifies the file to which to write the script.

-all

Writes the default values in addition to the current values of the variables.

-only_changed_vars

Writes only the changed variables. This is the default when no options are specified.

pattern

Writes the variables that match the specified *pattern*. The default is "*".

DESCRIPTION

The **write_app_var** command generates a Tcl script to set all application variables to their current values. By default, variables set to their default values are not included in the script. You can force the default values to be included by specifying the **-all** option.

EXAMPLES

The following is an example of the **write_app_var** command:

```
prompt> write_app_var -output sh_settings.tcl sh*
```

SEE ALSO

```
get_app_var(2)
report_app_var(2)
set_app_var(2)
```

write_container

Saves the information in the current or specified container to a file.

SYNTAX

```
write_container
   [ -container container_name | -r | -i ]
   [ -quiet ]
   [ -replace ]
   [-pre_set_top]
   filename
```

Data Types

```
container_name string
filename string
```

ARGUMENTS

-container *containerID*

Specifies the container to save. For information about how to specify a container ID, see the *Formality User Guide*.

-r

Saves the default reference container.

-i

Saves the default implementation container.

-replace

Replaces an existing file.

-quiet

Suppresses information and warning messages when writing to read-only containers.

-pre_set_top

Writes a container before a successfull **set_top** command.

filename

Specifies the file in which the tool saves container information.

DESCRIPTION

This command saves a container to a single file.

When you write a container to a file, you save only the design data and not not the setup parameters and verification results. The tool creates a file with the .fsc extension.

After you save a container, you can read it back in by using the **read_container** command.

When you use the *-pre_set_top* option, the container is written to a file before successful execution of the **set_top** command and stores intermediate data. These containers are not marked as read only.

Saved containers are marked as read-only when they are read into a session. All operations that would change the container contents are disallowed. For more information, see the **read_container** command man page. Therefore, you should fully prepare the containers for verification, including setting the top-level design before executing the **write_container** command.

The **write_container** command returns one of the following:

- 0 to indicate failure
- 1 to indicate success

EXAMPLES

The following example saves the design data for container *impl* into the file *impl_container* in the current working directory:

```
fm_shell> write_container -container impl impl_container
Info:  Wrote file 'impl_container.fsc'.
1
fm_shell>
```

SEE ALSO

```
read_container(2)
```

write_eco_regions

Writes the ECO region group and data files.

SYNTAX

```
status write_eco_regions
   [ -replace ]
   [ filename ]
```

Data Types

```
      filename string
```

Enabled Shell Modes

Match
Verify

LICENSE

This command is available when the "Formality-ECO" license key is used.

ARGUMENTS

-replace

Overwrites output files if they already exist.

filename

Specifies the names of the output files. If not specified, the default filenames "fm_eco_region.data.tcl" and "fm_eco_region.group.tcl" are used.

DESCRIPTION

This command can only be issued in match or verify mode. You must first issue the match_eco_regions command. The write_eco_regions command will then write the ECO region group and data files. The group file contains the previously identified ECO regions and their boundaries in the ECO reference design. The data file contains the ECO region boundary matching information between ECO reference and original implementation.

EXAMPLES

The following example will write the ECO region group and data files.

```
fm_shell (match)> match_eco_regions
fm_shell (match)> write_eco_regions -replace
```

SEE ALSO

```
match_eco_regions
create_eco_patch
set_orig_reference
set_orig_implementation
set_eco_reference
set_eco_implementation
```

write_edits

Writes the recorded edit commands to the specified file.

SYNTAX

```
write_edits
   filename
   [ -replace ]
```

Data Types

```
filename string
```

Enabled Shell Modes

Setup
Match
Verify

LICENSE

This command is available when the "Formality-Ultra" license key is used.

ARGUMENTS

filename

Specifies the name of the Tcl script file to write.

-replace

Replaces an existing file.

DESCRIPTION

This command writes the recorded edit commands (such as **create_net** and **disconnect_net**) to a Tcl file, which can then be used in Design Compiler or IC Compiler to replicate the edits.

Use the **record_edits** command to enable and disable recording.

The edit commands are written to a file except those that occurred while recording was turned off and ones that are reverted using the **undo_edits** command.

This Tcl script generated by this command contains two global variables that can be modified.

- **FM_ECO_ECHO_COMMANDS**
- **FM_ECO_ROOT_PATH**

You can set the **FM_ECO_ECHO_COMMANDS** variable to either 1 (enable) or 0 (disable). When the variable is enabled, the script does not execute design edit commands. Instead, it will echo these to the terminal. When the **FM_ECO_ECHO_COMMANDS** variable is set to 0, the design edit commands are executed. This is the default for the variable.

Set the **FM_ECO_ROOT_PATH** variable to the path of the design in which modifications will be made. This allows you to use the edits even if the design hierarchy in Formality and Design Compiler or IC Compiler do not match. For example, if the hierarchy in Formality is */top/m1/b1*, but you only read in the *b1* design in IC Compiler, set the **FM_ECO_ROOT_PATH** variable to *top/m1*.

Set the **FM_ECO_ROOT_PATH** variable to the empty string ("") if you read in the entire design. The empty string is the default value for the variable.

Two or more edit Tcl scripts can be concatenated together into a single script. Set the **FM_ECO_ECHO_COMMANDS** and **FM_ECO_ROOT_PATH** variables at the top of the combined file.

If you specifically do not write the recorded edits, Formality writes them to a file named *default_edits.tcl*.

EXAMPLES

Write the edit script to disk:

```
fm_shell (setup)> write_edits edits.tcl
```

SEE ALSO

```
create_net(2)
disconnect_net(2)
record_edits(2)
remove_net(2)
report_edits(2)
```

write_functional_matches

Writes functional matches as user-defined matches.

SYNTAX

```
write_functional_matches
   [-filename]
   file
```

Data Types

```
   file string
```

Enabled Shell Modes

Match Verify

ARGUMENTS

-filename

Required argument for a file.

file

The output file to generate the user matches from functional matches.

DESCRIPTION

This command writes the matches made by signature analysis (functional matches) as set_user_matches in the specified file.

The **write_functional_matches** command generates one line of output for each functional match. The line includes set_user_match, type, polarity, ref-name and impl-name.

EXAMPLES

The following example for write_functional_matches.

```
fm_shell (match)> write_functional_matches -f user_matches
fm_shell (match)>

An example of the file "user_matches" is as follows:
set_user_match -type cell -inverted \
 RTL:/WORK/rs_j0b1regs/j0b1_10_change_reg/updata_i_reg[0] \
 i:/WORK/rs_j0b1regs_f_p/j0b1_10_change_reg_updata_i_ff_b0
set_user_match -type cell -noninverted \
 RTL:/WORK/rs_j0b1regs/j0b1_10_change_reg/updata_i_reg[1] \
 i:/WORK/rs_j0b1regs_f_p/j0b1_10_change_reg_updata_i_ff_b1
```

write_hierarchical_verification_script

Writes a Tcl script that performs hierarchical verification on the current reference and implementation designs.

SYNTAX

```
status write_hierarchical_verification_script
   [-replace]
   [-noconstant]
   [-noequivalence]
   [-match type]
   [-save_mode mode]
   [-save_directory pathname]
   [-save_file_limit integer]
   [-save_time_limit integer]
   [-level integer]
   [-path instance-specific-pathname(s)]
   [-block instance-specific-pathname(s)]
   [-dont_resolve_failures]
   [-top_level_only]
   filename
```

Data Types

```
type string
mode string
pathname string
integer integer
filename string
```

ARGUMENTS

-replace

Replaces the existing script file.

-noconstant

Does not capture constant constraints at the boundary of hierarchical blocks verified in isolation.

-noequivalence

Does not capture equivalence constraints at the boundary of hierarchical blocks verified in isolation.

-match *type*

Writes matching commands for all objects, when *all* is specified, or only for objects matched by means other than name, when *auto* is specified. Specify one of the following for *type*:

- `auto (default)`
- `all`

-save_mode *mode*

Specifies the verification status for which a session file will be saved. In "auto" mode, Formality will save session files for inconclusive verifications only. If the "-dont_resolve_failures" switch is also specified Formality will save session files for failing and inconclusive verifications. Specify one of the following for *mode*:

- auto (default)
- not_passed
- failed
- inconclusive

-save_directory *pathname*

Saves the session files to the directory named *pathname*. The default is ".".

-save_file_limit *integer*

Specifies the maximum number of session files to save. The value must be greater than or equal to zero. The default is 1.

-save_time_limit *integer*

Saves session files using at least the specified number of cpu seconds. The specified integer must be greater than or equal to zero. Default: 0.

-level *integer*

Creates black boxes of the blocks only at or above this level, where top is level 0. This option also separately verifies the blocks unless used with the **-top_level_only** option. The specified integer must be greater than or equal to zero.

-block ***instance-specific-pathname(s)***

Creates black boxes of the specified blocks. This option also separately verifies the blocks unless used with the **-top_level_only** option.

-path ***instance-specific-pathname(s)***

Creates black boxes of the blocks along the specified instance specific path name. This option also separately verifies the blocks unless used with the **-top_level_only** option.

-top_level_only

Verifies only the top-level design with all blocks black boxed, except when used with the **-level**, **-block**, or -**path** options, in which case only the specified blocks are black boxed.

-dont_resolve_failures

Does not attempt to resolve failing lower-level hierarchical blocks.

filename

Specifies the script file name.

DESCRIPTION

This command writes a Tcl script that you can source to perform a hierarchical (block-by-block) verification on the current reference and implementation designs, where matched blocks are explicitly verified in isolation from containing blocks and without considering the function of contained (underlying) matched blocks. By default, the tool performs hierarchical verification for all matched blocks. You can specify which blocks to perform hierarchical verification on, by using the **-level**, **-block**, **-path** options.

The generated Tcl script performs as follows for each specified matched block of the currently defined top-level reference and implementation designs:

- Black-boxes specified matched lower level blocks.
- Removes unused compare points.
- Optionally captures matching information.
- Optionally captures constants at block boundaries.
- Optionally captures equivalences at block boundaries.

- `Verifies the target block as a top-level design.`
- `Saves the Formality session for failing/inconclusive verifications.`

Each block is verified in isolation, but explicit setup commands are generated to capture top-level context. By default, the tool attempts to resolve failing hierarchical blocks by reverifying them as part of their parent block. You can turn off this behavior by using the **-dont_resolve_failures** option.

The script records the SUCCEEDED, FAILED, or INCONCLUSIVE results for each block in a text file named fm_*script*.log. When the script terminates, the tool concatenates this file to the transcript.

The tool saves session files in a directory of your choice. You can control exactly which session files are saved by using the -save_mode option.

If either valid matching or verification results, or both, are available when the command is issued, the tool reuses them. Otherwise, it creates them before generating the script.

EXAMPLES

The following example writes a hierarchical verification script to a file named *my_script*.tcl in the current directory, replacing the existing file if it exists, and including all matching commands for all object types.

```
fm_shell> write_hi -rep my_script -match all
Status:  Checking designs...
Status:  Building verification models...
Status:  Matching...

************Matching Results******************
 144 Compare points matched by name
 24 Compare points matched by signature analysis
 0 Compare points matched by topology
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
**********************************************

Status:  Writing hierarchical verification script to file my_script.tcl...
```

SEE ALSO

```
verify(2)
set_parameter(2)
```

write_library_debug_scripts

Debugs failing cells from library verification mode in the Formality environment.

SYNTAX

```
write_library_debug_scripts
   [ -dir directory name ]
```

Data Types

```
      directory name string
```

ARGUMENTS

-dir *directory name*

Specifies the name of directory in which to save the Tcl scripts. Can be absolute or relative path. The default directory is $(pwd)/DEBUG.

DESCRIPTION

This command debugs a library cell that has failed/aborted verification in the standard Formality flow. The command generates Tcl scripts for failing cells. These run scripts can be invoked in a different Formality session and debugged using standard Formality diagnose commands.

EXAMPLES

```
fm_shell> write_library_debug_scripts -dir ./my_debug
Writing tcl script "./my_debug/test_my_failing_cell.tcl"
Generated "./my_debug/test_<cell_name>.tcl" scripts for all failing cells
Run 'formality -f <tcl_file>' to debug failing cells
```

To debug my_failing_cell run:

```
<unix_shell>% formality -f ./my_debug/test_my_failing_cell.tcl
```

SEE ALSO

```
verify(2)
select_cell_list(2)
report_cell_list(2)
```

write_power_model

Creates a Formality power model for a container's top design and saves it to a file.

SYNTAX

```
status write_power_model
   [ -container container_name | -r | -i ]
   [ -replace ]
   filename
```

Data Types

```
container_name string
filename string
```

ARGUMENTS

-container *containerID*

Specifies the container.

-r

Specifies the default reference container.

-i

Specifies the default implementation container.

-replace

Replaces an existing file.

filename

Specifies the file in which the tool saves the top level power model.

DESCRIPTION

The **write_power_model** command creates and saves the power model for the top design to a file. You can read in the the power model using the **read_power_model** command.

EXAMPLES

The following example creates and saves the power model for the top design in the container *impl* into the file *add2.fpm* in the current working directory.

```
fm_shell> write_power_model -container impl add2
Info:  Created and wrote model for top design 'add2'.
1
```

SEE ALSO

```
read_power_model(2)
```

write_register_mapping

Create and populate register mapping information file

SYNTAX

```
integer write_register_mapping
   [-rtlname]
   [-bbpin]
   [-bbox_input]
   [-bbox_output]
   [-bbox_inout]
   [-port]
   [-port_input]
   [-port_output]
   [-port_inout]
   [-unmatched_ref]
   [-siloti]
   [-replace]
   file_name
```

Data Types

file_name string

Enabled Shell Modes

Match Verify

ARGUMENTS

-rtlname

Enable RTL name instead of inferred register name

-replace

Overwrites the existing file

-bbpin

Include blackbox pin mapping in output report

-bbox_input

Include blackbox input pin mapping in output

-bbox_output

Include blackbox output pin mapping in output

-bbox_inout

Include blackbox inout pin mapping in output

-port

Include port mapping in output report

-port_input

Include input port mapping in output

-port_output

Include output port mapping in output

-port_inout

Include inout port mapping in output

-unmatched_ref

Include unmatched reference points in output

-siloti

Specify Siloti format for register mapping file

file_name

Output File Name

DESCRIPTION

The **write_register_mapping** command generates a report that maps registers, and optionally, ports and black-box pins in the reference design to the matching elements in the implementation design.

If the reference design is a RTL, the report is based on the origianl RTL names as the inferred register names might change because of SVF guidance related to register optimizations, such as merge, duplication, inv_push, constant optimization, or change_name. This makes it difficult to track inferred register names in the post SVF reference container. Therefore, this command generates the report by mapping the original register names to the matching registers in the implementation design.

The **-rtlname** switch enables Formality front end reader inferred register name to be repalced with RTL name. This is applicable to both reference and implementation container inferred register names.

The **-bbpin** switch enables including a list of all mapped black-box pins between the reference and implementation designs in the output report.

The **-bbox_input** switch enables including a list of all mapped black-box input pins between the reference and implementation designs in the output report.

The **-bbox_output** switch enables including a list of all mapped black-box output pins between the reference and implementation designs in the output report.

The **-bbox_inout** switch enables including a list of all mapped black-box inout pins between the reference and implementation designs in the output report.

The **-port** switch enables including a list of all mapped ports between the reference and implementation designs in the output report.

The **-port_input** switch enables including a list of all mapped input ports between the reference and implementation designs in the output report.

The **-port_output** switch enables including a list of all mapped output ports between the reference and implementation designs in the output report.

The **-port_inout** switch enables including a list of all mapped inout ports between the reference and implementation designs in the output report.

The **-unmatched_ref** switch enables including a list of all unmatched reference points in the output report.

The **-siloti** switch specifies Siloti format for output register mapping file.

EXAMPLES

The following command generates the report by mapping the original RTL registers names that are constant, optimized, or matched with the register names in the implementation design.

```
prompt>  write_register_mapping  Map.txt
```

Here are sample of what’s written out for various optimizations:

Below mapping shows RTL *reg o1* inferred *o1_reg* mapped to netlist libcell *o1_reg* having functionally read *Q* and *QN* driver pins. Match polarity is positive for *Q* pin and negative for *QN*.

```
oref pos u1/o1_reg
impl pos u1/o1_reg/Q
impl neg u1/o1_reg/QN
```

Below mapping shows above scenario with *-rtlname* switch enabled.

```
oref pos u1/o1
impl pos u1/o1_reg/Q
impl neg u1/o1_reg/QN
```

Below mapping shows unaltered instantiated tech libcell *inst_FD1* of tech library cell. *inst_FD1/QN* is missing as its functionally unread.

```
oref pos inst_FD1/Q
```

```
   impl pos inst_FD1/Q
```

Below mapping shows constant register optimization: *P_ref* is const0 optimized and *R_ref* is const1 optimized.

```
   oref pos mid/bot/P_reg
   impl c=0

   oref pos mid/bot/R_reg
   impl c=1
```

Below mapping shows register merge optimization: *o1_reg* and *o2_reg* are merged to *o1_reg*. *Q* and *QN* both pins are functionally read points.

```
   oref pos u3/o2_reg
   oref pos u3/o1_reg
   impl pos u3/o1_reg/Q
   impl neg u3/o1_reg/QN
```

Below mapping shows register duplication along with inv push:

```
   oref pos mid/bot/R_reg
   impl pos mid/bot/R1_reg/Q
   impl neg mid/bot/R2_reg/Q
   impl pos mid/bot/R2_reg/QN
```

Below mapping shows banked registers of two registers inside mapping along with change name guidance:

```
   oref pos mid/bot/q_reg[0]
   impl pos mid/bot/q_reg_1_0_/Q0

   oref pos mid/bot/q_reg[1]
   impl pos mid/bot/q_reg_1_0_/Q1

   oref pos mid/bot/q_reg[2]
   impl pos mid/bot/q_reg_3_2_/Q0

   oref pos mid/bot/q_reg[3]
   impl pos mid/bot/q_reg_3_2_/Q1
```

Below mapping shows 1:N user matching for instantiated libcells:

```
   oref pos FLOP2_reg/Q
   oref neg FLOP2_reg/QN
   impl pos FLOP2_reg/Q
   impl neg FLOP2_reg/QN
   impl pos FLOP3_reg/Q
   impl neg FLOP3_reg/QN
```

Below mapping shows retention register mapping post change name guidance. For *data_reg_0_/QN* is missing as its unread.

```
   oref pos fr2/regout_reg[3]
   impl pos fr2_regout_reg_3_/Q
   impl neg fr2_regout_reg_3_/QN

   oref pos data_reg_0_/Q
   impl pos data_reg_0_/Q
```

Below mapping shows *g1/q* having differnet polaity path case:

```
   oref  ?  g1/q
   oref neg g1/qn
```

```
impl  ?  g1/q
impl neg g1/qn
```

Below mapping shows unsupported retimed registers:

```
# The following mapping is not supported
# oref pos u_fcpu/fm_ret_fwmc_1_1_158/R_17725
# impl pos fm_ret_fwmc_1_1_158/R_17725
```

Below mapping shows a list of black-box pins (-bbpin) and ports (-port):

```
#
# Blackbox Pins
#
oref pos m1/bb1/in1
impl pos m1/bb1/in1

oref pos m1/bb1/in2
impl pos m1/bb1/in2

#
# Ports
#
oref pos in1
impl pos in1

oref pos out
impl pos out

oref pos q
impl pos q
```

Below mapping shows a list of registers with -siloti option:

```
#
# Registers
#
u1_reg/Q => u1

u1_reg/QN ~> u1

u2_reg/Q => u2

u2_reg/QN ~> u2
```

Below mapping shows a list of black-box output pins (-bbox_output) and input ports (-port_input) with -siloti option:

```
#
# Blackbox Pins
#
ram16x2/Q[0] => uram16x2/Q[0]

ram16x2/Q[1] => uram16x2/Q[1]

#
# Ports
#
clk_t => clk_t

rst => rst

sel1 => sel1
```

```
      sel2 => sel2
```

SEE ALSO

```
report_matched_points(2)
report_unmatched_points(2)
```