cadence®

System Connectivity Manager TCL Commands

Product Version 23.1

September 2023

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

1

Using TCL Commands

System Connectivity Manager supports TCL commands for performing various design tasks. However, to use the TCL commands you first need to launch the TCL shell.

Opening TCL Shell

On Windows

1. Launch SCM.
2. Choose *File – Open TCL Shell*.

   Use this shell to enter TCL commands for performing design tasks.
3. Any error or warning message thrown by SCM when a command is executed gets displayed in the TCL shell window.

On UNIX

On Unix platforms, the option to open the TCL shell needs to be specified when you launch System Connectivity Manager from the command line.

- At the command prompt, type the following command.

```
scm -tclshell
```

or

```
scm -proj <project_name> -tclshell
```

Important

Unless `-tclshell` option is specified, the default shell does not become the tcl shell.

The projectTCL.tcl File

From the SPB 16.01 release onwards, all tasks performed in one session of System Connectivity Manager, gets recorded in the `projectTCL.tcl` file. This file captures the commands specified in the tcl shell, as well as the actions performed using the System Connectivity Manager user interface. Any messages thrown by SCM during the design process are also captured in the `projectTCL.tcl` file. This file is saved in the `temp` directory, under the project directory.

Sourcing a TCL File

You can use the projectTCL.tcl file to recreate another instance of same design. The syntax used is:

`scm -tclFile <path to the .tcl file>`

This will create a set of design files that are in the same state as defined by the specified `.tcl` file.

TCL Syntax

- Syntax for all TCL commands is:

  `command_name arg1 arg2 arg3`

  The first word is the command name and rest are the arguments passed to that command.

- All commands, including the ones listed in this chapter, are case-sensitive.

- Use of forward slash and backslash is supported for specifying path. However, while using backslash in path names, you need to specify two backslash.

  For example, to specify the path as `D:\trial\abc.cpm`, the value to be entered in TCL is `D:\\trial\\abc.cpm`

- Square brackets do not indicate optional parameters.

- Square brackets followed by the TCL keyword, `list`, indicates that the information included with in the square braces is a list of related arguments.

  Example - `addSignal [list b1 inout 5]`

- Optional parameters are indicated using italic font.

- If value contains a space, enclose the value with double quotes or braces. For example,

```
addComponent ic_memory eeprom_8p_1 chips EEPROM_8P_1 -n 1 -k [list
PART_NAME=EEPROM_8P_1 DENSITY=256Kb SPEED=0.9uS "CONFIG=32K X 8"
TYPE=24C256 PART_NUMBER=1819-0115 PACK_TYPE=SO8 PTF=A
JEDEC_TYPE=SO00003 ]
```

Keywords in SCM Commands

When using tcl commands in System Connectivity Manager, there are a few keywords that have a special significance irrespective of the commands they are used in. Following table lists some of the keywords that are used in multiple commands.

Keyword	Indicates...	Example
inst	TCL command is valid for component instance	copy `inst i5` selectObject `inst i1`
net	TCL command is valid for a signal	delete `net add` selectObject `net add`
pin	TCL command is valid for a component pin or a port on a block	selectObject `pin [list i1 ao]`
dp	The following argument is for a differential pair signal	copy `net [list dp DP_data]` selectObject `net [list dp DP_A0`

The command specific keywords are explained along with the command syntax.

Help for TCL commands

For each TCL command, you can view the Usage String as well as the Command Help.

Usage String

A usage string is the command syntax without any explanation of the arguments passed to the command. To display the usage string for a TCL command, type the following in TCL shell and press Enter.

<command_name> -help

The usage string for the command gets displayed in the TCL shell itself.

Command Help

The command helps provides complete details about the specified command, such as common description, usage string, brief explanation of the command arguments, and examples if any.

The syntax to display the help for a command is:

`help <command_name>`

When you use the `help` command, the help opens in the CDNSHelp window.

2

Commands

addBlock

Adds an instance of a table, a Verilog, or a schematic block from the specified library to the design

Syntax

```
addBlock <library> <cell> <view> -n <count> -s |-p <string> | -r [list <start range>
     <end range>] | -o -inc <increment> -g [list <original global signal name> <new
     global signal name>] -reuse <reuse instance name>
```

where

library	Design library in which the block is saved
cell	Name of the block to be instantiated
view	Indicates the block type. The valid values are: ■ `tbl_1` ■ `sch_1` ■ `vlog_structural`
-n	Keyword used to indicate the number of instances of the block to be added to the design
count	Number of instances of the block to be added to the design
Block Packaging Options	
-s	Keyword to indicate that a suffix should be used to generate the reference designator for the block
-p	Keyword to indicate that a prefix should be used to generate the reference designator for the block

string	The string that is used as the suffix or the prefix value. This parameter is required only when either `-s` or `-p` is specified.
-r	Keyword to indicate that the reference designator for all the components in the block must fall within the specified range
start range	The reference designator range for the block
end range	
-o	Keyword used when the reference designators for the components in the block are to be preserved
`-inc <increment>`	Valid only when `-s` or `-p` options are used and when the value of count is greater than 1. The value by which the reference designator value for the first block are incremented, to generate the reference designator values for other instances of the same block.
`-g [list <original global signal name> <new global signal name>]`	keyword used to alias a global signal in the block to a signal in the design in which you are adding the block.
`-reuse <reuse instance name>`	Optional parameter, used if the block is to be used as a reuse block

Example

Command	Result
`addBlock tutorial_lib adder tbl_1 -r [list 77 99] -n 1 -inc 1`	An instance of spreadsheet block `adder` from the `tutorial_lib` library gets added to the design. The reference designator range specified for the components instantiated in the block is from 77 to 99.

Command	Result
`addBlock tutorial_lib sample tbl_1 -p TRY_ -n 1 -inc 1`	Adds an instance of the `sample` block from the `tutorial_lib` library to the design. When you open the sample block in context of the root design, reference designators for all the components will start with the prefix `TRY_`
`addBlock tutorial_lib analog_io sch_1 -o -n 1 -inc 1 -reuse ANALOG_IO_1`	Adds an instance of the schematic block, `analog_io`, from the `tutorial_lib` library to the design.

See Also

editBlock

createBlock

addBypass

Adds an instance of a bypass capacitor from the specified library to the component selected in the Component List pane.

For this command to run successfully, ensure that you have selected a component from the Component List pane.

Syntax

```
addbypass [-p/-parent <parent instance>] <library> <cell> <view>
     [physical_part_name] -net [list pinname1 signal1 pinname2 signal2] [-n
     number_of_instances ] [-value <property for value>] -k [list key_prop=value]
     -i [list injected_prop=value]
```

where

`-p/ -parent <parent instance>`	Keyword used to indicate the parent component.
library	Library name from which the bypass component is to be picked.
`cell`	Name of the part or the bypass component to be instantiated.
`view`	View name from which the component is to be instantiated. The valid values is: ■ `sym_1`: Instantiates the component as a symbol
`physical_part_name`	Optional parameter, indicating the physical part name of the part
`-net`	Keyword used to indicate which signal to connect to which pin of the bypass capacitor.
pinname1	The name of the first pin.
signal1	The name of the first signal.
pinname2	The name of the second pin.
signal2	The name of the second signal.
`-n`	Keyword used to denote the number of instances to be added
`number_of_instances`	Number of instances to be added.

`-value <property for value>`	Specifies the name of the property in the part table which is to be used to denote the value of the capacitor.
`-k [list key_prop=value]`	Specifies the name of the key property to be used to denote the value of the capacitor.
`-i [list injected_prop=value]`	keyword used to specify injected property values for the bypass.

Example

To add one instance of a bypass capacitor from the classlib to the current component:

```
addBypass -p inst i5 classlib cap sym_1 ACT574 -net[list vcc1 vcc gnd1 gnd]
```

Menu Text

Right click– Add Bypass Capacitors

addComment

Adds user comments on the specified instance, pin, or net.

Syntax

```
addComment <object_type> <comments> <object_name>
```

where

object_type Specifies the type of design object on which comment is to be added. The valid values are:

- inst

  Used if the comment is to be added on a component instance.

- net

  Used if the comment is to be added on a signal in the design.

- pin

  Used if the comment is to be added on a component pin.

Comment Specify the actual comment to be added. If the comment is more than a word, enclose it in double quotation marks.

object_name Indicates the design object on which comment is to be added.

In case the object type is `inst`, the valid values will be instance names, such as `i1`, `i2`, and so on.

For the `net` type object, logical signal name is specified as the second argument.

In case of `pins`, specify the pin name as the third argument.

However, in case a component has multiple pins with same pin name, both pin name and pin number, are specified to identify the pin. The syntax used is:

```
addComment pin comment [list pin_name pin_number]
```

Note: If `object_name` is not specified, the comment is added to the select object of the specified object type.

Example

Command	Result
`addComment`	Launches the Add Comment dialog box on the design object currently selected.
`addComment inst "This is a trial" i3`	Adds the comment "This is a trial" on component instance `i3`
`addComment net "Global signal from CPU block" address`	Adds the comment "Global signal from CPU block" on the `address` signal.
`addComment pin "A differential pin" din1 din2`	Adds the comment "A differential pin" on pin names `din1` and `din2`
`addComment net "Comment added to a bus" add<4..0>`	Adds the comment "Comment added to a bus" on to the vector signal `add<4..0>` **Note:** Comments added to a bus are not displayed on the individual signal bits.
`addComment net "Diff Pair signal" [list dp din1]`	Adds the comment "Diff pair signal" to a differential pair signal

Menu Command

Design – Comments – Insert Comments

See Also

modifyComment

deleteComment

addComponent

Adds the specified component to the design.

Syntax

`addComponent lib cell view` *physpartname -n count -k [list propname = propvalue] -i [list propname propvalue]*

where

`lib`	Library name from which the component is to be picked.
`cell`	Name of the part or the component to be instantiated.
`view`	View name from which the component is to be instantiated. The valid values are: ■ `sym_1`: Instantiates the component as a symbol ■ `chips`: Instantiates the component as a package ■ `tbl_1`: Instantiates a table block.
`physpartname`	Optional parameter, indicating the physical part name of the part
`-n`	Keyword used to denote the number of instances to be added
`count`	Number of component instances to be added.
`-k | -key`	Keyword used to indicate the key properties of the component in the corresponding part table file (`.ptf`)
`-i`	Keyword used to indicate the injected properties of the component in the corresponding part table file (`.ptf`)
`propname`	Name of the key or the injected property
`propvalue`	Value assigned to the property specified by the `propname` variable

Example

The command for adding one instance of component `act574` from the `classlib` library to the current design is:

```
addComponent classlib act574 chips ACT574 -n 1 -key [list PACK_TYPE=SOIC
     PART_NAME=ACT574 ]
```

```
addComponent ic_memory eeprom_8p_1 chips EEPROM_8P_1 -n 1 -k [list
PART_NAME=EEPROM_8P_1 DENSITY=256Kb SPEED=0.9uS "CONFIG=32K X 8"
TYPE=24C256 PART_NUMBER=1819-0115 PACK_TYPE=SO8 PTF=A
JEDEC_TYPE=SO00003 ]
```

Menu Text

Design – Add Component

See Also

- replaceComp
- modifyComponent

addConnection

Captures the connectivity between the specified signal name and the component pin.

Syntax

```
addConnection <signal name> [list <instance name> <pin name> <pin number>]
addConnection [list <instance name> <pin number> <net msb> <net lsb> <net name>]
```

where

`signal name`	Name of the signal
`instance name`	Instance name
`pin name`	pin name for which the pin-net connectivity is to be defined
`pin number`	pin number that is to be connected to the specified signal.
net msb	Most significant bit of the net
net lsb	Least significant bit of the net
net name	net name of which the connection is to be specified

Note: The second syntax is valid when Signal Connectivity Details pane is the active window.

Example

Command	Result
`addConnection dclk [list i10 clk 11]`	Connects net `dclk` to pin `11` of component instance `i10`.
`addConnection ba<7> [list i10 d1 3]`	Connects net `ba<7>` to pin `3` of component instance `i10`.

Command	Result
`addConnection [list i19 7 -1 -1 BNC1]`	Connects the scalar signal BNC1 to pin 7 of instance i19. **Note:** This command format works only when the signal connectivity pane in the active window

`selectObject net brd`
`editConnectivity`
`selectWindow ccp`
`addConnection [list i3 36 ]`
	Connects the scalar signal `BNC1` to pin 36 of instance i3.
	
	Note: This command format works only when the signal connectivity pane in the active window

See Also

deleteConnection

addProperty

Adds a property to the selected instance.

For this command to work properly, you must select the Properties window, and the property definition must exist in the Project settings.

Syntax

```
addProperty -prop [list <property name> <property value>]
```

where

`-prop`	Keyword to indicate an operation on a property.
`<property name>`	The name of the property to be added to the selected instance.
`<property value>`	Value of the property

Examples

Command	Result
`addProperty -prop[list COMP_NAME LED]`	Adds the property `COMP_NAME` and assigns the value `LED` to it.
`addProperty -prop[list COMMENT sample_comment]`	Adds the property `COMMENT` and assigns the value `sample_comment` to it.

Menu Command

Right click on the Property window and choose *Insert Property*

See Also

changeProperty

deleteProperty

addPupd

Adds a Pull up/Pull Down to the selected net.

Syntax

```
addPupd -type <pullup/pulldown> -lib <library> -cell <cell> -view <view> -physname
     <physical part name> -k [list key_prop=value] -i [list injected_prop=value] -
     value <property for value> -net <signal name> [-common]
```

where

`-type pullup/pulldown`	
`-lib <library>`	Library name from which the component is to be picked.
`-cell <cell>`	Name of the part or the component to be instantiated.
`-view <view>`	View name from which the component is to be instantiated. The valid values are: ■ `sym_1`: Instantiates the component as a symbol ■ `chips`: Instantiates the component as a package ■ `tbl_1`: Instantiates a table block.
`-physname <physical part name>`	The physical part name of the component
`-k [list key_prop=value]`	Specifies the name of the key property to be used to denote the value of the pullup/ pulldown.
`-i [list injected_prop=value]`	keyword used to specify injected property values for the pullup/pulldown.
`-net <signal name>`	The signal to pullup or pull down to.
`[-common]`	

Examples

Command	Result
`addPupd -type Pulldown -lib discrete -cell resd -view sym_1 -physname RESD -k [list RATED_POWER=1/8W PKG=1206 PART_NAME=RESD TOL=5% VALUE=10M ] -value VALUE -net GND -parent BNC1`	Adds a pulldown to the selected signal.

Menu Command

Select a signal from the Signal list pane and from the right-click menu choose *Add Pullup/ Pulldown*

addSignal

Adds signals to the open design. The signals added are listed in the Signal List pane.

Syntax

```
addSignal [list <signal_name> <signal_scope> <voltage> <differential pair name>]
```

where

`signal_name`	Logical name of the signal to be added to the design
`signal_scope`	Specify the scope of the signal to be added to the design. The valid values are: ■ input ■ output ■ inout ■ global ■ local
`voltage`	Specifies the voltage value to be assigned to the signal being added. **Note:** This is an optional parameter that is to be added only if a voltage value is to be assigned to the signal.
`differential_pair_name`	Specifies the name of the differential pair signal to be added. This optional parameter is to be specified only when you want to a differential pair signal to the design.

Examples

Command	Result
`addSignal [list add input] [list a1 inout]`	Following signals are added to the design. ■ add ■ a1
`addSignal [list data inout "" dp_data]`	Differential pair signal `dp_data`, with member signals as `data+` and `data-` are added to the design
`addSignal [list b1 inout 5]`	Signal `b1` with the voltage value 5 V is added to the design.
`addSignal [list ab<1..0> inout]`	Adds a 2-bit bus with scope set to INOUT.

Menu Command

Design – Add Signal

See Also

delete

addConnection

aliasSignal

addTermination

Adds termination to a given pin.

Syntax

```
addTermination -type <terminaton type> -part<n> -lib <library name> -cell <cell
     name> -view <view name> -physname <physical name> -k [list <property
     name>=<property value> -i [list <property name>=<property value> -low <low
     voltage signal> -high <high voltage signal> -delay <delay constraint>
```

where

`-type`	Indicates the type of termination
`-part`	Indicates the part to be selected for termination - eg if a termination has two resistors then part1 identifies the first resistor and part2 identifies second resistor
`-lib`	Library name from which the component is to be picked.
`-cell`	Name or part of the component to which termination is added.
`-view`	View name from which the component is to be instantiated
`-physname`	Optional parameter, indicating the physical part name
`-k`	Keyword used to indicate the key properties of the component in the corresponding part table file
`-i`	Keyword used to indicate the injected properties of the component in the corresponding part table file
`-low`	Keyword to indicate the low voltage signal value
`-high`	-Keyword to indicate the high voltage signal value
`delay`	Keyword to indicate the delay constraint

Example

```
addTermination -type SeriesCapacitor -part1 -lib discrete -cell "cap_np" -view
     sym_1 -physname CAP_NP -k [list "PACK_TYPE=SMDCAP" "PART_NAME=CAP_NP"
     "VALUE=.01uf" "TOLERANCE=+20%/-80%"]  -value "VALUE"  -delay "0.1 ns"
```

Menu Command

Object – Associated Components– Add Termination

addTool

Adds the specified command name to the toolbar menu.

Syntax

`addTool user|tcl <menu text> <command name|tcl function name>` *-proj -mps -dir <intial directory> <arguments>*

where

`user | tcl`	Keywords used to indicate whether a user-defined or a TCL command is to be added to the *Tools* menu
`menu text`	String that appears in the Tools drop-down menu
`command name | tcl function`	The path to the user tool (`.exe` or `.com` files) or the name of the TCL command
`-proj`	Optional parameter to be used only in case of user-defined tools. When specified, the project path is passed as an argument to the executable
`-mps`	Optional parameter to be used only in case of user-defined tools. When specified, the mps session is passed as an argument to the user tool
`-dir`	Keyword used to specify the initial directory
`intial directory`	Path to the initial directory
`arguments`	Arguments that are to be passed to the command or to the TCL function

Examples

Command	Result
`addTool`	Opens the Add Tools dialog box
`addTool user notepad C:\WINDOWS\system32\notepad.exe`	Adds a new menu command notepad to the `Tools` drop-down menu in System Connectivity Manager

Menu Text

Tools – Add Tools

See Also

launchTool

aliasSignal

Creates an alias of the specified signal names.

Syntax

For creating an alias of scalar signals:

```
aliasSignal first_net_name second_net_name
```

For creating an alias of vectored signals:

```
aliasSignal [list <first_net_name> <lsb> <msb> <step>] [list <second_net_name>
     <lsb> <msb> <step>]
```

where

`first_net_name`	Names of the signals to be aliased
`second_net_name`	
`lsb`	Least significant bit
`msb`	Most significant bit
`step`	An optional parameter The step value is used to calculate the next bit to be aliased. By default, when this parameter is not specified, the step value is 1.

Examples

Command	Result
`aliasSignal aa bb`	Aliases net `aa` to net `bb`
`aliasSignal [list a 4 8] [list b 0 4]`	Creates an alias between highest 5 bits of vector signal `a<8..0>` and the bits of signal `b<4..0>`

Command	Result
`aliasSignal [list a 0 6 2] [list b 0 3]`	Aliases following signal bits. ■ `a<0>` aliased to `b<0>` ■ `a<2>` aliased to `b<1>` ■ a<4> aliased to `b<2>` ■ `a<6>` aliased to `b<3>`

Menu Text

Object – Alias

See Also

- unAliasSignal

ascend

Ascends the design hierarchy by opening the parent block for the current block.

Syntax

```
ascend
```

Example

Command	Result
`ascend`	Ascends the hierarchical block currently selected

Menu Text

Design – Ascend

assignModel

Assigns the specified model to the component and component pins.

Syntax

```
assignModel inst|pin <device_model> instance_name|pin_details
```

where

`inst`	keyword used to indicate that the command is used to assign device model to a component instance
`pin`	keyword used to assign device model to a component pin
`device_model`	Name of the model in the `.dml` file
`instance name`	Name of the instance to which the model is assigned. **Note:** Specified when `inst` is used as the second argument
`pin details`	Details of the component pin to which the model is assigned. The pin details are specified using the following format: `[list instance_name pinname pin_number]` **Note:** Specified when inst is used as the second argument

Examples

Command	Result
`assignModel`	Invokes the SI Model Assignment dialog box
`assignModel inst DS90C031TM i19`	Assigns the model, `DS90C031TM`, to the component instance `i19`.

Command	Result
`assignModel pin DS90C031TM_DIN [list i12 din1 1]`	Assigns the model, `DS90C031TM`, to the component pins din1 of component instance `i12`.

Menu Text

Object – SI Model – Assign Model

See Also

- removeModel

assignPinNumber

Exchanges the location of two pins across two functions in a multi-function component by swapping two pins.

Syntax

```
assignPinNumber <pin number>
```

where `<pin number>` refers to the pin to be swapped with the currently selected pin.

For this command to run successfully, ensure that a pin is selected in the Component Connectivity Details pane.

Example

Command	Result
`assignPinNumber 11`	Swap pin 11 with the currently selected pin.

Menu Text

Right Click – Assign Pin Number – More.

assignPower

Modifies the existing power and NC pins assignment of a component. Using this command you can switch the pin types between NC and Power.

Tip

This command is valid only for global pins that have the POWER_PINS property assigned to them.

For this command to run successfully, ensure that the Component List pane is selected.

Syntax

```
assignPower [list instances] [list powersignal|NC pinnumbers pinname]
```

where

`[list instances]`	Instance name(s) for which the assignment of the power and NC pins of a component is modified.
`powersignal | NC`	Name of the global power signal to be assigned to the specified pin. **Note:** While specifying the name of the power signal ensure that the signal scope is Global and the physical net name is specified.
`pinnumbers`	Pin number of the global power pin for which power assignment is to be modified.
`pinname`	Optional parameter Pin name for which the power assignment is to be modified.

Example

Command	Result
`assignPower`	Assign Power dialog box is launched on the component currently selected.

Command	Result
`assignPower i5`	Assign Power dialog box is launched on the specified instance.
`assignPower [list i4 i5] [list NC 25]`	Pin 25 of instance `i4` and `i5` becomes an NC pin.
`assignPower [list i4 i5] [list AGND 9]`	Pin 9 of instance `i4` and `i5` gets connected to the power signal AGND.
`assignPower [list i5] [list AGND 9] [list NC 24] [list 12V 25]`	Following pins of instance `i5` are modified: ■ Pin 9 is assigned the global signal AGND ■ Pin 24 is marked as an NC pin ■ Pin 25 is assigned the global signal 12V

Menu Text

1. Select a component in the Component List pane.
2. Choose *Design – Assign Power*

assignPower

Modifies the existing power and NC pins assignment of a component. Using this command you can switch the pin types between NC and Power.

Tip

This command is valid only for global pins that have the POWER_PINS property assigned to them.

For this command to run successfully, ensure that the Component List pane is selected.

Syntax

```
assignPower [list instances] [list powersignal|NC pinnumbers pinname]
```

where

`[list instances]`	Instance name(s) for which the assignment of the power and NC pins of a component is modified.
`powersignal | NC`	Name of the global power signal to be assigned to the specified pin. **Note:** While specifying the name of the power signal ensure that the signal scope is Global and the physical net name is specified.
`pinnumbers`	Pin number of the global power pin for which power assignment is to be modified.
`pinname`	Optional parameter Pin name for which the power assignment is to be modified.

Example

Command	Result
`assignPower`	Assign Power dialog box is launched on the component currently selected.

Command	Result
`assignPower i5`	Assign Power dialog box is launched on the specified instance.
`assignPower [list i4 i5] [list NC 25]`	Pin 25 of instance `i4` and `i5` becomes an NC pin.
`assignPower [list i4 i5] [list AGND 9]`	Pin 9 of instance `i4` and `i5` gets connected to the power signal AGND.
`assignPower [list i5] [list AGND 9] [list NC 24] [list 12V 25]`	Following pins of instance `i5` are modified: ■ Pin 9 is assigned the global signal AGND ■ Pin 24 is marked as an NC pin ■ Pin 25 is assigned the global signal 12V

Menu Text

1. Select a component in the Component List pane.
2. Choose *Design – Assign Power*

assocCompViewSelectTab

Selects a particular tab window in the associated component viewer.

Syntax

```
assocCompViewSelectTab bypass/term/pupd
```

where

`bypass`	Opens the bypass capacitor tab.
`term`	Opens the termination tab.
`pupd`	Opens the pullup/pulldown tab.

Menu Text

Object – Associated Components – Edit Bypass Capacitor

baseline

This command baselines a design. In case other designers have imported the design as a read-only block SCM notifies them that the design has been baselined, and they can update the imported blocks.

Syntax

```
baseline -major/-minor/-custom [-version <version>] [-comment <comment string>] [-
     all] [-default]
```

where

`-major/ -minor / -custom`	Keyword to specify the type of baseline version. For example, if the current version number is 2.0, baselining the design as a major version will increment the version number to 3.0. while baselining the design as a minor version will increment the version number to 2.1. Use custom to specify your own values as specified by the -*version* parameter.
`-version <version>`	In case of a custom baseline, this optional keyword is used to specify the version number.
`-comment <comment string>`	Optional keyword to add a comment to the baselined design.
`-all`	Optional parameter to baseline all the sub-blocks used in the current design.
`-default`	In case the custom version number provided is incorrect, the tool suppresses any error messages, specifies a valid version number and baselines the design.

Example

Command	Result
`baseline -major -comment "Updated with universal Clk signals" -all`	Baselines the design as a major version and adds a comment to the baselined design.
`baseline -custom -version 6.7.0 -all`	Baselines the design with a custom version 6.7.0.

Menu Text

Choose *Design – Baseline Design.*

blockRefdesRange

Use to specify or remove the reference designator range to be used for components in a block.

Syntax

```
blockRefdesRange -remove <from> <to>
```

where

`-remove`	Use this option to remove the
`<from> <to>`	New reference designator range

Example

Command	Result
`blockRefdesRange`	Launches the *Edit Design Ref Des Range* dialog box
`blockRefdesRange -remove`	Removes any reference designator range specified for the block
`blockRefdesRange 55 100`	Updates the reference designator range for the current block, such that while packaging the design the reference designators used for components in the block lie between 55 to 100. If the reference designators for the existing components is U1, U2, U3 and so on, these will get modified to U55, U56, U57 and so on.

Menu Text

Design – Edit Block Ref Des Range

changeCustomColumn

Changes the values of the custom column for the selected object in the Component List pane, Signal List pane, or the Component Connectivity Details pane.

For this command to work properly, ensure that a you have added the custom column and selected an object.

Syntax

```
changeCustomColumn -col <custom column name> -val <new value>
```

where

`-col`	Keyword to indicate an operation on a custom column.
`<custom column name>`	The name of the custom column to be changed.
`-val`	Keyword to indicate the new value
`<new value>`	New value to be updated.

Examples

Command	Result
`changeCustomColumn -col TOLERANCE -val 2`	Changes the value in column TOLERANCE to 2 for the selected component.

changePhysname

Assigns the specified name as the new physical net name.

Note: Command not valid for vectored signals or buses.

Syntax

```
changePhysname <new_pnn>
changePhysname [list <net_name> <new_pnn>][list dp <diffpair name> <new_pnn>]
```

where

`new_pnn`	The physical net name to be assigned to the selected or the specified net
`net_name` `diffpair_name`	Existing logical name of a regular or a differential pair signal.

Example

Command	Result
`changePhysname PAR`	Changes the physical net name of the currently selected net to PAR
`changePhysname [list fpga FGA] [list add ADDRESS]`	Changes the physical net name of the `fpga` and `add` nets to `FGA` and `ADDRESS`, respectively. **Note:** The net name specified after the `list` keyword is the logical net name.
`changePhysname [list dp DP1 DIFF]`	Changes the physical net name of the differential pair signal to `DIFF`. **Note:** On changing the physical net name of a differential pair, the logical name of the differential pair signal also changes.

Menu Text

Object – Change – Physical Net Name

changeProperty

Changes a property value of the selected instance.

For this command to work properly, you must select the Properties window, and the property must be attached to the component.

Syntax

```
changeProperty -prop [list <property name> <new property value>]
```

where

`-prop`	Keyword to indicate an operation on a property.
`<property name>`	The name of the property to be added. Ensure that a definition for the property exists in the project settings.
`<new property value>`	New value of the property

Examples

Command	Result
`changeProperty -prop[list COMP_NAME LED]`	Changes the value of property `COMP_NAME` and assigns the value `LED` to it.
`changeProperty -prop[list COMMENT sample_comment]`	Adds the property `COMMENT` and assigns the value `sample_comment` to it.

See Also

addProperty

deleteProperty

changeRefdes

Changes the existing reference designator of a component to the one specified by the user.

Syntax

```
changeRefdes <new_reference_designator>

changeRefdes [list <instance name> <new reference designator>]

changeRefdes tool
```

where

`instance name`	Component instance for which the reference designator value is to be modified.
`new_reference_designator`	Reference designator to be assigned to the specified component or to the component currently selected in the Component List pane
`tool`	Modifies the reference designator value assigned by the tool **Note:** Corresponding to the menu text, *Design – Change – Ref Des – Tool Assigned*

Example

Command	Result
`changeRefdes J15`	Changes the reference designator of the selected component to `J15`.
`changeRefdes [list i7 U44]`	Changes the reference designator of instance i7 to U44.
`changeRefdes [list i2 J7] [list i8 U20]`	Changes the reference designator of instance i2 to J7 and instance i8 to U20.

Menu Text

Object – Change – Ref Des – User Assigned

changeRefdesAssoc

Changes the refdes of an associated component throught the Assoc Comp Window.

Syntax

```
changeRefdesAssoc [list <old reference designator> <new reference designator>]
```

where

`old reference designator`	Name of the old reference designator that needs to be changed.
`new reference designator`	Keyword to defines the new name for the new reference designator that is added

EMenu Text

Object – Change – Ref Des

changeRoot

Opens the specified block in the master mode

Syntax

```
changeRoot libname cellname viewname
```

where

`libname`	Name of the library in which the block to be opened is saved
`cellname`	Name of the block to be opened in master mode.
`viewname`	View to be opened. The valid values are: ■ `tbl_1`: for a table block ■ `vlog_structural`: for a Verilog block

Example

Command	Result
`changeRoot`	Launches the Change Root dialog box.
`changeRoot test_lib data tbl_1`	Makes the table block, `data` as the root design and open it in the master mode.
`changeRoot test_lib data vlog_structural`	Opens the Verilog block, `data` in the master mode.

Menu Text

Project – Change Root

changeScope

Changes the scope of the signal

Syntax

```
changeScope <scope>
changeScope [list <net name> <scope>]
changeScope| [list dp <diffpair_name> <scope>]
```

where

`net name` — Signal name for which the scope is to be modified.

If signal name is not specified, the scope of the signal currently selected in Signal List pane (slp) is modified

`scope` — New scope of the signal. Possible values are:

- input
- inout
- output
- local
- global

Example

Command	Result
`changeScope global`	Changes the scope of the selected signal to Global.
`changeScope [list wstat global]`	Converts `wstat` to a Global signal.
`changeScope [list DP_signal input]`	Converts the scope of the member nets of DP_signal to a input signal.

Menu Text

Object – Change – Signal Scope

changeVoltage

Assigns or modifies the value of the VOLTAGE property on a signal. For this command to work ensure that the Signal List pane is active or selected.

Syntax

```
changeVoltage [list <net name> <voltage value>]
```

where

`net name`	Signal name for which the voltage value is to be modified
`voltage value`	Voltage value to be assigned to the signal

Example

Command	Result
`changeVoltage`	Launches the DC Voltage dialog box on the selected net
`changeVoltage [list dout 5]`	Assigns a voltage of 5V to the `dout` signal
`changeVoltage 25mV`	Assigns a voltage of 25mV to the signal currently selected in the Signal List pane
`changeVoltage [list a<2..0> 2]`	Assigns a voltage of 2V to the vectored signal `a<2..0>`

Menu Text

Object – Change – DC Voltage

See Also

selectWindow

closeAllWindows

Closes all open windows. You can save the data and close the windows or close the windows without saving the data.

Syntax

```
closeAllWindows -save
```

where

`-save`	Saves the data in the open windows.

Example

Command	Result
`closeAllWindows -save`	Saves all data in the open windows and then closes the windows.
`closeAllWindows`	Closes all windows without saving.

closeDesign

Closes the current design.

Syntax

```
closeDesign
```

Menu Text

Choose *File – Close*

closeShell

Closes the current TCL Shell.

Syntax

```
closeShell
```

Menu Text Choose *View – Close TCL Shell*

cmReplay

Plays a Constraint Manager script.

Syntax

cmreplay <CM script file>

Example

Command	Result
cmReplay /home/usr/ Documents/action.scr	On Unix, runs the Constraint Manager script `action.scr` located in directory */home/usr/ Documents*
cmReplay d:\\work\\action.scr	On Windows, runs the Constraint Manager script `action.scr` located in directory *d:\work*

Menu Text

1. Choose *Design – Edit Constraints*. Constraint Manager window opens.
2. In Constraint Manager, choose *File – PlayBack Script.*

copy

Copies the selected or the specified design object.

Syntax

```
copy inst <object_name>
copy net <object_name> | [list dp <diff_pair_name>]
copy pin [list <inst_name> <pinname> <pin_number>]
```

where

`inst`	Keyword used if a component instance is to be copied
`net`	Keyword used to copy a signal
`pin`	Keyword used to copy a pin
`<object_name>`	Depending on whether the previous argument is `inst`, `net`, or `pin` the `object_name` is an instance name, signal name, or pin name, respectively.
`[list ... ]`	TCL keyword
`dp`	Keyword used to copy a differential pair signal
`<diff_pair_name>`	Name of the differential pair to be copied

Example

Command	Result
`copy`	Copies the design object currently selected. It can be a components, a net, a pin, or the test currently selected.
`copy inst i5`	Copies instance `i5`
`copy net data`	Copies the signal named `data`
`copy net [list dp DP_data]`	Copies the differential pair signal named `DP_data`
`copy pin [list i7 ce* 24]`	Copies pin number 24 of instance `i7`.

Menu Text

Edit – Copy

See Also

paste

createBlock

Creates a block in the specified library

Syntax

```
createBlock <library name> <block name> <view> [list <port name> <port type>
     <diffpair name>] -inheritglobals [list <signal name>] -refdesrange <range
     start> <range end>
```

where

`library name`	Name of the library in which a block is to be created.
`block name`	Name of the block to be created
`view`	Specifies the block type. The valid values are: ■ tbl_1: create a spreadsheet block ■ sch_1: creates a schematic block ■ `vlog_structural`: creates a Verilog block
`[list <port name> <port type> <diffpair name>]`	List of ports to be added to the block *port name:* name of the port *port type:* valid values are; IN, INOUT, OUT *diffpair name:* name of the differential pair - required only if the port is added as a differential pair port
`-inheritglobals`	Keyword used to include the existing GLOBAL signals in the block being created
`[list <signal name>]`	Name of the GLOBAL signals to be included in the block
`-refdesrange`	keyword used to define a range as the valid reference designator range for the components in a block.
`<range start> <range end>`	start and the end values for the reference designator range

Example

Command	Result
`createBlock tutorial_lib hadder tbl_1 [list bwr OUT] [list dclk IN] -inheritglobals [list GND VCC] -refdesrange 50 100`	Creates a table block, `hadder`, in the `tutorial_lib` library. The block has two ports, an `OUT` port `bwr`, and an `IN` port `dclk`, defined. The global signals in the root design, GND and VCC were added as ports in the new block. All the components instantiated in the `hadder` block will have reference designators values between 50 and 100.
`createBlock tutorial_lib chill tbl_1 [list aa IN aa] [list bb INOUT]`	Creates a table block, `chill`, in the `tutorial_lib` library.
`createBlock tutorial_lib halfadd vlog_structural`	A Verilog block, `halfadd`, is created in the `tutorial_lib`

Menu Text

Project – Create Block

Design– Create Block

createDiffPair

Creates a user-defined differential pair, using two existing signals.

Select the Signal List Pane before you run this command.

Syntax

```
createDiffPair <first net name> <second net name>
```

Example

Command	Result
createDiffPair Clock+ Clock-	Creates a differential pair called `Sig_DFClock` comprising the signals `Clock+` and `Clock-`.

Menu Text

1. Select the two signals to create a Differential pair.
2. Right-click and choose *Create Differential Pair*.

createSubProject

Create a new project using the settings of the design currently open in System Connectivity Manager. The libraries available for the new project is same as the libraries available for the design currently open in System Connectivity Manager.

Syntax

```
createSubProject <proj location> <proj name> <library name> <block name> <view>
     [list <port name> <port type> <diffpair name>]
```

where

`proj location`	Location in which the project is to be created.
`proj name`	Name of the project to be created. The project file created is `projname.cpm`.
`library name`	Working library for the sub-project to be created.
`block name`	Name of the root design. This block is created in the working library specified by `library name`.
`view`	Indicates the type of the block. The valid values are `tbl_1` for a spreadsheet block and `verilog` for a Verilog block.
`[list <port name> <port type> <diffpair name>]`	Optional parameter Specifies the port name and the port type added to the block.

Example

Command	Result
`createSubProject d:/dessamples/modules/subproj projname blcklib blckname tbl_1 [list po1 IN] [list po2 IN]`	Creates a SCM project, projname in the `d:/dessamples/modules/subproj` folder. The project has a spreadsheet block, `blckname` in the `blcklib` library. The block has two input ports `po1` and `po2`.

Menu Text

Project – Create Sub-Project

cut

Removes the specified design object from the design and saves it on the clipboard.

Syntax

```
cut inst|net <object_name>
```

where

`inst|net` — Keywords used to identify the design object to be cut.

To cut a component instance, use the `inst` keyword. To cut a standard or a differential pair signal use the `net` keyword.

`<object_name>` — Name of the design object to be cut. Valid values are instance name, signal name, or differential pair.

Note: The details for a differential pair are specified using the TCL keyword `list`.

```
[list dp <diff_pair_name>]
```

Example

Command	Result
`cut`	Removes the selected component
`cut inst i5`	Cuts instance `i5`
`cut net VCC`	Removes the VCC net from the design and puts it on the clipboard, provided VCC is an unconnected net.
`cut net [list dp DP_data]`	Cuts the differential pair signal named `DP_data`

Menu Text

Edit – Cut

See Also

paste

delete

Removes the specified design object.

Syntax

```
delete inst|net <object_name> | [list dp <diffpair net name>]
```

where

`inst`	Keyword used if a component instance is to be copied
`net`	Keyword used to copy a signal
`<object_name>`	Name of an instance or a signal depending on whether the keyword specified is `inst` or `net`, respectively.
`[list dp <diff_pair_name>]`	Used with `net` keyword, to delete a differential pair signal specified by `diff_pair_name`

Note: Only unconnected signals can be deleted from a design. Before deleting a net, remove its connectivity by using the deleteConnection command.

Example

Command	Result
`delete`	Removes the selected component or net
`delete inst i5`	Deletes instance `i5 from the des`
`delete net [list dp DP_add]`	Removes the differential pair signal named `DP_add`

Menu Text

Edit – Delete

See Also

deleteConnection

deleteBypass

Deletes a Bypass Capacitor.

For this command to work successfully, select the component from which the Bypass capacitor is to be deleted.

Syntax

```
deleteBypass <refdes of bypass capacitor> -parent <parent instance name>
```

where

`<refdes of bypass capacitor>`	The Reference designator of the bypass capacitor to be deleted.
`-parent <parent instance name>`	The parent instance from which the bypass capacitor is to be deleted.

Example

Command	Result
`deleteBypass C1`	Deletes the bypass capacitor C1 from the currently selected component.
`deleteBypass C2 -parent i1`	Deletes the bypass capacitor C2 from parent component i1.

deleteComment

Deletes the comments added to the specified instance, net, or pin.

Syntax

```
deleteComment <object_type> <object name>
```

where

object_type — Specifies the type of design object from which comment is to be deleted. The valid values are:

- inst

  Used if the comment is to be deleted from a component instance.

- net

  Used if the comment is to be deleted from a signal in the design.

- pin

  Used if the comment is to be deleted from a component pin.

object_name — Indicates the design object from which comment is to be deleted.

For `inst` object type, the valid values are instance names, such as `i1`, `i2`, and `i9`.

For the `net` type object, `object_name` is the logical signal name.

In case of `pins`, specify the pin name. However, in case a component has multiple pins with same pin name, both pin name and pin number, are specified to identify the pin.

Note: If `object_name` is not specified, the comment is deleted from the first object of the specified object type.

Example

Command	Result
`deleteComment net b1`	Deletes comment added to the `b1` signal.
`deleteComment inst`	Deletes the comment attached to the instance currently selected in the Component List pane
`deleteComment`	Deletes the comment attached to the currently selected design object. The object can be a net, an instance, or a pin.
`deleteComment pin ce*`	Deletes the comment from pin `ce*`. **Note:** For this command to work, the Component Connectivity Details window should be the active window.

Menu Command

Design – Comments – Delete Comments

See Also

addComment

modifyComment

deleteConnection

Removes the connectivity for the specified pin.

Syntax

```
deleteConnection [list <instance name> <pin name> <pin number>]
```

where

instance_name	Component instance from which the connected signals are to be removed
pin name	Name of the component pin from which the signals are to be removed
pin number	Pin number from which the signals are to be removed

Example

Command	Result
`deleteConnection [list i1 ce* 6]`	Removes the connectivity and all the associated components for pin `ce*` of instance `i1`.
`deleteConnection [list i1 ce* 6] [list i2 reset 27]`	Removes connectivity for pin `ce*` of instance `i1` and pin `reset` of instance `i2`

See Also

addConnection

deleteTermination

Removes the termination(s) attached to the specified component pin(s). For this command to work, ensure that the Component Connectivity Detail pane (ccp) is selected.

Syntax

```
deleteTermination
deleteTermination [list <instance name> <pin name> <pin number>]
```

where

instance_name	Component instance from which the terminations are to be removed
pin name	Name of the component pin from which the terminations are to be removed
pin number	Pin number from which the terminations are to be removed

Example

Command	Result
`deleteTermination`	Removes terminations attached to the selected pins in the Signal Connectivity pane.
`deleteTermination [list i3 add14 57] [list i3 add15 67]`	Removes terminations attached to pin 57 and 67 of instance `i3`.
`deleteTermination [list i1 6] [list i2 11]`	Removes terminations attached to pin 6 of instance i1 and pin 11 of instance `i2`.

See Also

deletePupd

deleteProperty

Deletes a property from the selected instance.

For this command to work properly, you must select the Properties window, and the property must be attached to the instance.

Syntax

```
deleteProperty -prop <property name>
```

where

`-prop`	Keyword to indicate an operation on a property.
`<property name>`	The name of the property to be added. Ensure that a definition for the property exists in the project settings.

Examples

Command	Result
`deleteProperty -prop COMP_NAME`	Deletes the property `COMP_NAME` from the selected instance.

See Also

addProperty

changeProperty

deletePupd

Removes the pull up or pull down resistor(s) from the specified nets. For this command to work, ensure that the Component Connectivity Detail pane or Signal List pane is selected.

Syntax

```
deletePupd up|down [list <instance name> <pin name> <pin number>]
```

where

up down	Keywords used to indicate whether the pull up or the pull down resistor is to be removed.
instance_name	Component instance from which the pullup or pulldown resistors are to be removed
pin name	Optional Parameter Name of the component pin from which the pullup or pulldown resistors are to be removed
pin number	Pin number from which the pullup or pulldown resistors are to be removed

Example

Command	Result
`deletePupd up [list i1 ce* 6]`	Removes the pullup resistor from pin 6 of instance i1
`deletePupd down [list i1 27]`	Removes the pulldown resistor from pin 27 of instance i1
`deletePupd down [list i2 ce* 6] [list i2 23]`	Removes the pulldown resistors from pin 6 and pin 23 of instance i2

See Also

deleteTermination

descend

Opens the specified block in context of the root design.

Syntax

```
descend instancename
```

where

instancename	Instance of the block to be opened in context mode

Example

Command	Result
`descend`	Descends the hierarchical block currently selected
`descend i8`	Opens the block instance `i8` in context mode **Note:** For the descend command to work, the Component List pane (`clp`) should be selected.

Menu Text

Design – Descend

See Also

selectWindow

ascend

designComment

Adds, removes, and modifies comments added to the design currently opened in System Connectivity Manager.

Syntax

```
designComment add|delete|edit <comment>
```

where

`add`	keyword used to add a comment to the design
`delete`	keyword used to delete comment from the design
`edit`	keyword used to change the existing comment
comment	Comment to be added today

Example

Command	Result
`designComment`	Launches the Comments dialog box
`designComment add "This is a design comment"`	Adds the comment — *This is a design comment* — to the design opened in System Connectivity Manager
`designComment edit edited`	Replaces the existing design comment with the specified comment — *edited*. **Note:** If the comment is a single string then it need not be enclosed in quotation marks.
`designComment edit "Modified design comment"`	Replaces the existing design comment with the specified comment — `Modified design comment`.
`designComment delete`	Removes the existing comment from the design

Menu Text

Design – Comments – Insert Comment

Design – Comments – Delete Comment

Design – Comments – Edit Comment

dumpAssocComp

Dumps the contents of the Associated Components Viewer to a file.

For this command to work successfully, select the Associated Components Viewer window.

Syntax

dumpAssocComp <filename>

Example

Command	Result
`dumpAssocComp /home/usr/Documents/assoc.txt`	Writes the contents of the Associated Components Viewer to the file `assoc.txt`.

dumpGlobalNavigate

Dumps the contents of the Signal Navigate window to a file.

For this command to work successfully, select the Signal Navigate window.

Syntax

dumpGlobalNavigate <filename>

Example

Command	Result
`dumpGlobalNavigate /home/usr/Documents/signals.txt`	Writes the contents of the Signal Navigate window to the file `signals.txt`.

dumpGrid

Dumps the contents of the Component List, Signals List or Component Connectivity Details, to a text file. If the exists, then the contents are appended to the file.

For this command to work successfully, select the grid you want to export.

Syntax

dumpGrid <filename>

Example

Command	Result
`dumpGrid /home/usr/ Documents/grid.txt`	Writes the contents of the Signal Navigate window to the file `grid.txt`.

dumpProperty

Dumps the contents of the Property window, to a text file.

Syntax

dumpProperty <filename>

Example

Command	Result
`dumpProperty /home/usr/ Documents/props.txt`	Writes the contents of the Properties window to the file `props.txt`.

dumpVDD

Dumps the contents of the Visual Design Differences to a file.

For this command to work successfully, select the Visual Design Differences window.

Syntax

dumpVDD <filename>

Example

Command	Result
`dumpVDD /home/usr/ Documents/vdd.txt`	Writes the contents of the Visual Design Differences to the file `vdd.txt`.

editBlock

Opens the specified block for editing.

Syntax

```
editBlock libname cellname viewname
```

where

`libname`	Library in which the block is saved
`cellname`	Name of the block
`viewname`	Depending on the type of block to be edited, the valid values are: ■ tbl_1 ■ sch_1 ■ vlog_structural

Example

Command	Result
`editBlock`	Launches the Edit Block dialog box
`editBlock tutorial_lib adder tbl_1`	Open the spreadsheet block `adder`, from the `tutorial_lib` library in the master mode for editing
`editBlock tutorial_lib analog_io sch_1`	Opens the schematic block, `analog_io`, from the `tutorial_lib` library in Design Entry HDL.

Menu Text

Project – Edit Block

See Also

createBlock

editBypass

Edits a Bypass capacitor.

Syntax

editBypass <oldrefdes> <[-p/-parent <parent instance>] -refdes <new refdes> -dcnets [list <old signal 1> <new signal 1> <old signal 2> <new signal 2>] -k [list physical_prop=value] -i [list injected_prop=value]

where

`<old refdes>`	Indicates the old reference designator.
`-p/ -parent`	Keyword used to indicate the parent component.
-refdes <new refdes>	Keyword to indicate the new refDes, and the value for the new refdes.
`-dcnets [list <old signal1> <new signal1> <old signal2> <new signal2>]`	Keyword to indicate the list of old and new DC nets
`-k [list key_prop=value]`	The value by which the reference designator value for the first block are incremented, to generate the reference designator values for other instances of the same block.
`-i [list injected_prop=value]`	keyword used to inject property values to the bypass.

Example

Command	Result
`editBypass C4 -refdes C2`	Edits the bypass and changes the reference designator from C4 to C2.
`editBypass C2 -refdes C3 -dcnet [list VCC VCC_D GND GND_D]`	Edits the bypass capacitor C2, and changes the reference designator to C3, and modifies the DC net.

Menu Text

Right click – Edit Bypass Capacitors.

editConnectivity

Open the connectivity of the selected component instances in Component Connectivity Details pane and in case of signals, open the connectivity of selected signals in Signal Connectivity Details pane.

Syntax

```
editConnectivity -samepane
```

where

-samepane	Optional parameter When specified, opens the connectivity of all the selected components in the same pane of the Component Connectivity Details window.

Example

Example 1: A section the script that opens that connectivity of multiple components in the same pane of the Component Connectivity Details window.

```
...
selectObject inst i1 i2 i5
editConnectivity -samepane
...
...
```

Example 2: A section the script that opens three separate panes to display the connectivity of three component instances.

```
...
selectObject inst i1 i2 i5
editConnectivity
...
```

Example 2: A section the script that opens the connectivity of net `brd` in Signal Connectivity Details pane.

```
...
selectObject net brd
editConnectivity
...
```

Menu Text

Object – Edit Connectivity

Object – Edit Connectivity in Same Pane

editConstraints

Opens the Constraint Manager.

Syntax

editConstraints

Example

Command	Result
editConstraints	Opens Constraint Manager

Menu Text

Design – Edit Constraints

editLogicalPin

Switches the application to the logical edit mode and displays the Component Connectivity Details pane to edit or view the logical connections.

For this command to work properly, ensure that you are in the *Physical Part Connectivity Details* pane.

Syntax

```
editLogicalPin
```

Example

Command	Result
`editLogicalPin`	Switches to Logical view of the design.

Menu Text

View – Logical View

See Also

editPhysicalPin

editPhysicalPin

Switches the application to the physical edit mode and displays the *Physical Part Connectivity Details* pane to edit or view the physical connections.

For this command to work properly, ensure that you are in *Component Connectivity Details* pane.

Syntax

```
editPhysicalPin [list <instance name> <pin name> <pin nubmer>] -save/-nosave
```

where

`<instance name>`	Indicates the instance of the pin.
`<pin name>`	Indicates the pin name of the pin.
-<pin number>	Indicates the pin number
`-save/nosave`	Option to specify to save the design while switching to Physical view.

Example

Command	Result
`editPhysicalPin [list i1 clk 2] -save`	Edits the Pullup/pulldown connected to parent net BNC1 and changes the reference designator from R2 to R3.

Menu Text

View – Physical View

See Also

editLogicalPin

editPupd

Edits the Pullup/ Pulldown

Syntax

```
editPupd <reference designator> -refdes <new reference designator> -net <new signal
     name> -k [list <new property=value>] -i [list <new property=value>]
```

where

<reference designator>	The RefDes of the pullup/pulldown.
-refdes	Keyword to indicate the new reference designator
<new reference designator>	The new reference designator
-net	Keyword to indicate the net name to attach the Pupd
<new signal name>	The name of the new signal.
`-k [list key_prop=value ]`	Specifies the name of the key property to be used to denote the value of the capacitor.
`-i [list injected_prop= value]`	keyword used to specify injected property values for the bypass.

Example

Command	Result
`editPupd R2 -refdes R3 - parent BNC1`	Edits the Pullup/pulldown connected to parent net BNC1 and changes the reference designator from R2 to R3.

Menu Text

Object – Associate Components —Edit Constraints

editTerminations

Edits the Terminations attached to a pin.

For this command to work, select a pin that has a termination to edit.

Syntax

```
editTermination -oldrefdes <old reference designator> -newrefdes <new reference
     designator> -k [list <property name>=<new property value> -i [list <property
     name>=<new property value> -low <new low voltage signal> -high <new high
     voltage signal> -delay <new delay constraint>
```

where

-oldrefdes	Keyword to indicate the existing reference designator of the termination.
`<old reference designator>`	The existing reference designator of the termination.
-newrefdes	Keyword to indicate the new reference designator of the termination.
<new reference designator>	The new reference designator of the termination.
-k [list key_prop=value]	Specifies the name of the key property to be used to denote the value of the capacitor.
-i [list injected_prop=value]	keyword used to specify injected property values for the bypass.
-low	Keyword to indicate the new low voltage signal value.
<new low voltage signal>	The new low voltage signal value.
-high	Keyword to indicate the new high voltage signal value.
<new high voltage signal>	The new high voltage signal value.
-delay	Keyword to indicate the delay constraint.
`<new delay constraint>`	The new delay constraint.

Example

Command	Result
`editTermination`	Displays the Edit Termination dialog box.
`editTermination -oldrefdes R6 -k [list MATERIAL=EMPTY WATTAGE=200V PACK_TYPE=1206 PART_NAME=RES VALUE=0 TOLERANCE=1A ] -low gnd`	Edit the termination.

Menu Text

Right-click and choose *Edit Terminations*

editUserPins

Modifies the user pins on a co-design object. Select a co-design object in the Component pane to edit the user pins.

Note: The Edit User Pins command is available only for co-design objects. System Connectivity Manager when launched from the Cadence SiP Digital Architect GXL or XL products, provides support for co-design objects.

Syntax

```
editUserPins
```

Menu Text

Object – Edit User Pins

exit

Closes System Connectivity Manager

Syntax

```
exit save | nosave
```

where

save	.Saves the design before closing System Connectivity Manager
nosave	.Closes System Connectivity Manager without saving design files

Example

Command	Result
`exit`	Displays an message asking whether or not the files are to be saved before closing SCM.
`exit save`	Saves the design files and closes System Connectivity Manager.

Note: Using the `exit` command, before an openProject command will kill System Connectivity Manager without a saving.

Menu Text

File – Exit

expandPins

Expands and collapses the vectored pins in the Components Connectivity Details pane.

Syntax

```
expandPins on/off [-selection]
```

where

on/off	.Enables/ disables the *Expand All Pins* function.
-selection	.Optional parameter to change the scope of expand/ collapse function to the currently selected vectored pin.

Example

Command	Result
`expandPins on`	Expands all the vectored pins in the Components Connectivity Details pane.
`expandPins off`	Collapses all the vectored pins in the Components Connectivity Details pane.
`expandPins on -selection`	Expands the selected vectored pins in the Components Connectivity Details pane.

Menu Text

Component Connectivity Details pane – *Expand All Pins* check box

exportInterface

Exports the interface definition of packaged (BGA) component between PCB and SiP projects in an XML format

This command is supported if you are using System Connectivity Manager with one of the following licenses:

- SIP Digital Architect GXL
- SIP Digital Architect XL

Syntax

```
exportInterface -inst <instance> -file <xml file path>
```

where

-inst	.Keyword to define the instance of the component.
<instance>	.The instance name of the packaged component.
-file	.Keyword to indicate the file name where to export the interface.
<xml file path>	.The path and name of the XML file to store the interface.

Example

Command	Result
`exportInterface -inst U3 -file /home/documents/interface.xml`	Exports the interface definition to the XML file.

Menu Text

Design – Export Interface

exportPhysical

Creates design files required for creating the physical layout of the design either in Allegro PCB Editor or in SiP Layout.

Syntax

```
exportPhysical pcb|sip genpkg -i <input file> -o <output file> remove_etch
     ignore_fixed -p always|same|never overwrite|export updateboard -l
     Allegro_PCB_Editor|Allegro_PCB_SI|APD|CDNSIP
```

where

pcb sip	Keywords used to indicate whether the physical layout is for a PCB Board or for a SiP Package
`genpkg`	Keyword used to indicate that the package data is to be generated for the design.
-i	keyword used to indicate that the next parameter specifies the input board or sip layout file
`input file`	The board (`.brd`) or the sip (`.sip`) file to be used as the input file for creating the design in SCM **Note:** This is an optional parameter. Specifying an input file insures that the same setup information, as specified in the input file, is used to generate the physical design for the current project.
-o	keyword used to specify the output board or sip layout file.
`output file`	The board (`.brd`) or the sip (`.sip`) file that will have all the modifications made to the physical design. **Note:** If you have generated the board file at least once, the name of the output file is automatically seeded as the name of the input file.
`remove_etch`	When specified the connect lines can be modified
`ignore_fixed`	When specified, indicated that the components with the Fixed property are to be ignored while generating the physical layout file for the design.

`-p` Keyword used to indicate that the next argument specifies how to handle the placed parts during the ECO process.

`always|same|never`

Always

All components in the physical layout are replaced with the new parts from System Connectivity Manager according to their reference designators and at the same x/y location and rotation as the old component.

Same

Components in the physical layout are replaced with the new components provided that the package symbol, value, and the tolerance of the new component is same as that of the component being replaced. If the package data changes, the old component is removed from the physical layout, but the changed part is treated as an unplaced part.

Never

Components in the physical layout are not replaced automatically. You need to modify interactively.

`overwrite|export` Specifies how electrical constraints are to be handled during the ECO process.

Overwrite

Electrical constraints are completed removed from the physical layout file and then the constraints from the logical design are imported.

Note: Not recommended if you have made changes to the electrical constraints in the physical layout

export

Only the modified constraints are imported from the design in System Connectivity Manager to the physical layout.

`updateboard` Keyword used to indicate whether the physical layout is to be updated or not

`-l`	Keyword used to specify the tool to be launched for opening the layout design file
`Allegro_PCB_Editor|Allegro_PCB_SI|APD`	The tools in which the physical layout file is opened. ■ Allegro_PCB_Editor - Allegro PCB Editor ■ Allegro_PCB_SI - Allegro PCB Signal Integrity ■ APD - Allegro Package Designer ■ cdnsip - Cadence SiP Digital Layout ■ None - Do not launch any tool

Example

Command	Result
`exportPhysical pcb genpkg None -I -O -P never`	Only package files are generated for the design
`exportPhysical pcb genpkg updateBoard -I -O shilpa.brd -P never`	Package files are generated and the physical layout of the design is created in the `shilpa.brd` file.
`exportPhysical pcb genpkg updateBoard -O temp.brd ignore_fixed -P always export -L Allegro_PCB_Editor`	Creates the package files and a board file, temp.brd, for the current design in System Connectivity Manager. The temp.brd file is opened for viewing in Allegro PCB Editor.

Menu Text

Project – Export – PCB Board

Project – Export – SiP Package

See Also

importBlock

filterBlock

This command is valid only for the physical view. Specified the command to filter out the blocks that are now to be included in the physical view for the design.

Syntax

```
filterBlock exclude | -i <block hierarchical path>
```

where

`exclude`	Keyword used when you want to remove component instances from the physical view.
`-i`	Keyword used when you want to include components instances from hierarchical blocks in the physical view.
`block hierarchical path`	Path of the hierarchical block in context of the root design

Example

Command	Result
`filterBlock`	Opens the Physical View Block Selection dialog box
`filterBlock i10:i3`	Opens the instance i3 of the subblock, that is instantiated in the block instance i10 of the root design
`filterBlock exclude all`	Removes all instances of components that are instantiated in hierarchical blocks, from the physical view of the design.
`filterBlock -i all`	Displays all instances of components instantiated in hierarchical blocks in the physical view.
`filterBlock -i i10:i3`	Includes the components instantiated in the instance i3 of the subblock, which in turn is instantiated in the block instance i10 of the root design

Menu Text

Design – Filter Blocks

Note: Available in the physical view (*View – Physical View*)

filterRows

Filters the rows in the Component Connectivity Details and Physical Parts Connectivity Details panes.

Syntax

```
filterRows -col <column name> -filter <filter string>
```

where

`-col`	Keyword used to indicate the column name.
`<Column Name>`	The name of the column to filter by.
`-filter`	Keyword used to indicate the filter.
`-<filter string>`	The value to filter by.

Example

Command	Result
`filterRows -col Signal -filter DP_*`	Displays all rows where the signal name begins with `DP_`.

genVerilogNetlist

Generates the physical or logical verilog netlist.

Syntax

```
genVerilogNetlist physical/logical
```

where

`physical/logical`	Keyword to indicate type of Verilog Netlist.

Example

Command	Result
`genVerilogNetlist logical`	Generates a logical Verilog netlist.
`genVerilogNetlist physical`	Generates a physical Verilog netlist.

Menu Text

Project – Generate Verilog Netlist – Logical

Project – Generate Verilog Netlist – Physical

getProjectPath

Returns the path of the current project.

Syntax

```
D:\16.6\ade_Tutorial_Examples\modules\hier_design\hier_design\hier_design.cpm
```

globalFind

Displays the Global Find dialog box in which you specify a net or cell to be located in your design.

Syntax

```
globalFind -net/-n <net name> [-p [list <propertyName=PropertyValue>]]

globalFind -lib/-l <library name> -cell/-c <cell name> -view/-v <view name> [-p
     [list <property Name=Property Value>]]

globalFind -p <propertyName=PropertyValue>
```

where

`-net`	Indicates the name of the net that is searched.
`-p`	Indicates the name of the property of the net to be searched.
`-lib`	indicates the library from the component is to be searched
`-c`	-indicates the name of the component
`-view`	indicates the view of the component to be searched

Example

Command	Result
`globalFind -net/-n <BNC> [-p [list <Voltage=0>`	Displays all nets where the net name begins with BNC.
`globalFind -n abc*`	Displays all nets where the net name begins with ABC

globalReplace

Replaces instances of a component in the design with another component.

Syntax

```
globalReplace -oldnet/-oldn <search net name> [-oldp [list <property Name=Value>]]
     -newnet/-newn <net name> [-newp [list <property Name=Value>]]

gglobalReplace -oldlib <library name> -oldcell <cell name> -oldview <view name> [-
     oldp [list <property Name=Value>]] -newlib <library name> -newcell <cell name>
     -newview <view name> [-newp [list <property Name=Value>]]

[-k [list <key_prop=value>]] [-i [list <inj_prop=value>]] [-match both/pinname/
     pinnumber]

[-mapsig [list <src PinName> <src PinNumber> <new PinName> <new PinNumber>]]

[-unmapsig [list <src PinName> <src PinNumber> <new PinName> <new PinNumber>]]

[-mapprop [list <PropName> <propValue>]] [-unmapprop [list <PropName>
     <propValue>]]
```

where

`-oldnet`	Indicates the name of the net that needs to be replaces.
`-oldp`	Indicates the property value of the old net that needs to be replaced
`-newnet`	Indicates the name of the new net.
`-newp`	Indicates the new properties that are assigned to the new net.
`oldlib`	Indicates the name of the existing library that needs to be replaced
`oldcell`	Indicates the name of existing component that needs to be replaced
`oldp`	List of properties on the existing component that is being replaced
`oldview`	Indicates the name of existing component view that needs to be replaced
`newlib`	Indicates the name of the new library

`newcell`	Indicates the name of the new component
`newview`	Indicates the view name of the new component
`newp`	Indicates the property list of the new component
`-k`	Key property values assigned to the new components
`-i`	Injected property values assigned to the new components
`-mapsig`	Indicates which old pinname connectivity is to be mapped to which new pinname
`-unmapsig`	used to unmap the connectivity - required only when default mapping results in some pins being mapped which we do not wish to map
`-mapprop`	Indicates the properties of the old component that need to be retained on the new component.
`-unmapprop`	Indicates the properties that need to be removed from the new component

Example

Command	Result
`globalReplace -oldp test1=123 -newp test1=12345`	The old property value test123 is replaced with the new property value test1234
`globalReplace -oldnet abc -newnet cba`	Old net abc is replaced with new net cba

Command	Result
`globalReplace -oldlib global_replace_lib -oldcell 74138 -oldview chips -oldp [list LOCATION=D14 ] -newlib global_replace_lib -newcell 7400 -newview chips -k [list PART_NAME=74HC00 PART_NUMBER=NEW3040000 TRADE_CODE=74HC00 ] -match pinname`	Old library is replaced by new library

GNSelectBit

Selects a particular net bit in the signal navigate window.

Syntax

```
GNSelectBit -sig <signal name> -bit <bit to select>
```

where

`-sig`	Name of the signal.
`-bit`	Keyword used to indicate the net bit to be selected

help

Launches the help for the specified command in the CDNSHelp viewer.

Syntax

```
help <command_name>
```

Example

Command	Result
`help addSignal`	Opens the help for the `addSignal` command in the Cadence Help viewer.
`help importPhysical`	Opens the help for the `importPhysical` command in the Cadence Help viewer.

importBlock

Imports the specified schematic, table, or verilog block in the project currently open in System Connectivity Manager.

You can import a block either by specifying the project file (`*.cpm`) or by specifying the library file (`cds.lib`).

Syntax

```
importBlock CDS|CPM <file_path> lib cell view <import_mode>
```

where

CDS|CPM	Indicates whether the block data is to be read from the project file (`*.cpm`) or from the library file (`cds.lib`). The valid values are: ■ CDS: Indicates that the `cds.lib` file is to be imported ■ CPM: Indicates that the project file (`*.cpm`) is to be imported
`file_path`	Specifies the file to be used for importing the block. **Note:** If the first parameter is specified as CDS, then the second parameter should be the `cds.lib` path, else it is the path to the project file.
`lib`	Name of the design library that contains the block is to be imported
`cell`	Name of the block to be imported
`view`	View in which the block is to be imported.

import_mode

Specifies whether the block is to be imported as a read-only block or as a read-write block.

The valid values are:

- r: read-only block
- w: read-write block

Example

Command	Result
`importBlock CPM D:/analog_io/ analog_io.cpm analog_io_lib analog_io sch_1 w`	Imports the schematic block, `analog_io` as from the `analog_io_lib` library, into the current design library. The block is imported as a read-write block.
`importBlock CDS D:/design_samples/ subproj/cds.lib samllib adder tbl_1 r`	The `adder` block from the `smallib` library is imported in the worklib for the current design as a read-only table block

Menu Text

Project — Import — Block

importECONetlist

Imports the connectivity information from the netlist file into the design.

Syntax

`importECONetlist <netlist_file>`

where

`netlist_file`	Path to the netlist file to be imported.

Example

Command	Result
`importECONetlist`	Opens the Import ECO Netlist dialog box
`importECONetlist fpga.txt`	Updates the design with the connectivity data in the `fpga.txt` netlist file.

Menu Text

Project — Import — ECO Netlist

importPhysical

Imports the physical design data from the Allegro PCB Editor layout database to the logical design in System Connectivity Manager.

Syntax

```
importPhysical
importPhysical layout | feedback [preview] <filename>
```

where

layout	Use this keyword if the design in System Connectivity Manager is to be updated with the modifications in the layout file.
feedback	Use this keyword if the design in System Connectivity Manager is to be updated with the modifications in the packaged.
preview	Use this keyword to display the design differences in the Visual Design Differences window in SCM UI.
filename	Name of the board file (`*.brd`) if the layout files are being imported. Complete path of the `packaged` view if the feedback files are being imported in the logical design.

Example

Command	Result
`importPhysical`	Launches the Import Physical dialog box
`importPhysical layout preview temp.brd`	Lists all the design differences between the currently open project and specified `temp. brd` file, in the Visual Design Differences pane in SCM.

Command	Result
`importPhysical layout temp.brd`	Updates the logical design with all the differences between the currently open project and specified `temp. brd` file.

Menu Text

Project — Import — Physical

See Also

exportPhysical

importVerliog

Imports the specified Verilog file into design database

Syntax

`importVerilog <path of verilog file> <lib>` *-r -t -i <module name> -m <file name>*

where

path	Path to the Verilog file to be imported
library	Design library in which the block created by importing the verilog file is saved
`-r`	Optional When specified, this argument indicates that for all the components in the Verilog file, instance names will be used as reference designators
`-t`	Import the Verilog module as a table block
`-i`	Optional When specified, indicates that only interface data is to be imported for the module
`<module name>`	Optional The name of the module for which the interface data is to be imported
`-m`	Optional When specified, indicates that a map file is used to map the component names in the Verilog file to component names in the libraries added for the current project.
`<filename>`	Optional The name of the map file used for mapping

Example

Command	Result
`importVerliog`	Launches the Import Verilog dialog box
`importVerilog D:\\design_samples/ user_pins_added.v tutorial_lib -r -t`	All the modules in the Verilog file, `user_pins_added.v`, are imported to the `tutorial_lib` library as table blocks
`importVerilog D://verilog_import/ top.v tutorial_lib -t -i adc_top`	The interface data for the `adc_top` module in the `top.v` Verilog file is imported as a table block in the `tutorial_lib` file.

Menu Text

Project — Import — Verilog

launchTool

Launches the specified tool from System Connectivity Manager.

Syntax

`launchTool <toolname>`

where

toolname	The tool to be launched from System Connectivity Manager. The valid values are: ■ allegro_layout — Launches Allegro PCB Editor ■ allegro_si — Launches Allegro PCB SI ■ sip_layout — Launches SiP Layout ■ pdv — Launches Part Developer (Allegro PCB librarian) ■ mi — Allegro PCB Model Integrity

`launchTool -menu <menu_text>`

where

-menu	Keyword used for specifying menu text
menu text	The string that appears in the Tools drop-down menu for the user-defined applications or the TCL commands

Example

Command	Result
`launchTool` allegro_layout	Launches Allegro PCB Editor

Menu Text

Tools – Allegro PCB Editor

Tools – Allegro PCB SI

Tools – SiP Digital Layout

Tools – Signal Integrity – Allegro PCB MI

Tools – Allegro Part Developer

See Also

addTool

loadSignal

Includes the signals listed in the specified signal (`.sig`) file in the current design.

Syntax

```
loadSignal filepath
```

where

filepath	Path to the signal (`.sig`) file to be imported

Example

Command	Result
`loadSignal`	Launches the Open Signal File dialog box
`loadSignal d:/signal_lis.sig`	Adds the signals listed in the `signal_list.sig` file, to the design

Menu Text

Design – Load Predefined Signals

See Also

saveSignal

logicalView

Opens the design in the logical mode. This command is valid only when the design is open in the physical mode.

Syntax

```
logicalView -nosave
```

Menu Text

View – Logical View

See Also

physicalView

markAsCodesign

Marks the design as a co-design die.

The command markAsCodesign is only valid with the SIP license.

Syntax

```
markAsCodesign <instance name hierarchical path>
```

Menu Text

Right-click on a instance in the Hierarchy Viewer and choose *Mark As Codesign*.

modifyComment

Replaces the existing comment with the new comment. This command is valid only for design objects that have a user comment specified on it.

Syntax

```
modifyComment <object_type> <comments> <object name>
```

where

object_type	Keywords used to specify the type of design object on which comment is to be modified. The valid values are: ■ inst — Used when the comment on a component instance is to be modified. ■ net — Used while modifying comment on a signal. ■ pin — Used if the comment on a component pin is to be modified.
Comments	The comment to be added. If the comment is more than a word, enclose it in double quotation marks.
object_name	Design object on which comment is to be added.

Example

Command	Result
`modifyComment`	Launches the Add Comment dialog box on the design object currently selected in System Connectivity Manager.
`modifyComment inst "termination added" i5`	Modifies the existing comment on component instance `i5`, to *termination added*.
`selectObject inst i1` `modifyComment pin "Add 4 bypass caps" eo*`	The existing comment on pin `eo*` of instance `i1` is modified to *Add 4 bypass caps*.

Menu Text

Design – Comments – Edit Comments

See Also

addComment

deleteComment

modifyComponent

Modifies the physical properties of a component instance.

Syntax

```
modifyComponent <instance name> -k [list <property name> <property value>] -i [list
     <property name> <property value>] [-apply_on_excluded]
```

where

instance name	Name of the instance to be modified
-k	Keyword used to indicate the key properties of the component in the corresponding part table file (`.ptf`)
`<property name>`	Name of the property in the ptf file
`<property value>`	Value assigned to the specified property.
-i	Keyword used to indicate the injected properties of the component in the corresponding part table file (`.ptf`)
[-apply_on_excluded]	This parameter is used only if the active view is the physical view.

Example

```
modifyComponent -k [list PACK_TYPE SOIC] [list PART_NAME TC55B4257]
```

```
modifyComponent -k [list VOLTAGE 25V] [list MATERIAL CERM] [list PKG 0805] [list
     PART_NAME CAP] [list TOL 3] [list VALUE .01UF] -i [list JEDEC_TYPE 0805_T]
```

modifyCompAssoc

Modifies the Associated component.

Syntax

```
modifyCompAssoc <reference designator> -k [list <key property name> <property
     value>] -i [list <injected property name> <property value>]
```

where

<reference designator>	Reference designator of the associated component to be modified.
-k	Keyword used to indicate the key properties of the component in the corresponding part table file (`.ptf`)
`<property name>`	Name of the property in the ptf file
`<property value>`	Value assigned to the specified property.
-i	Keyword used to indicate the injected properties of the component in the corresponding part table file (`.ptf`)
`<injected property name>`	Name of the injected property.
`<property value>`	Value assigned to the injected property.

Example

```
modifyCompAssoc R11 -k[list comment=sampleComment]
```

openDesign

Opens the specified hierarchical block either in master mode or in context of the root design

Syntax

```
openDesign instance_name
openDesign -name <block name>
```

where

instance name	Instance of the hierarchical block to be opened in context of the root design
-name	Keywords used when the specified block — instantiated in the design — is to be opened in the master mode
block name	Name of the instantiated block to be opened in master mode

Example

Command	Result
`openDesign -name processor`	Opens the processor block in the master mode
`openDesign i15`	Opens the block instance `i15` in context of the root design **Note:** If `i15` is a component instance, an error message is thrown.
`openDesign i10:i3`	Opens the instance i3 of the subblock, that is instantiated in the block instance i10 of the root design

openFile

Opens the specified HTML or the design file in System Connectivity Manager.

Syntax

```
openFile <path_to_the_file>
```

where

path_to_the_file	Full path to the file to be opened, along with the file extension. **Note:** If backslash (\ ) is used in the pathname, you need to use two backslash (\\). This is required because as per the TCL syntax, backslash (\ ) is used to comment the character immediately after the \.

Example

Command	Results
`openFile`	Launches the *Open* dialog box. Use this dialog box to browse to the required `HTML` or `dsr` file.
`openFile D:/design/report.html`	Open the `report.html` file located in the `D:/design` folder.
`openFile D:\\design\\rep.dsr`	Opens the `rep.dsr` file in the `D:\design` folder.

Menu Text

File – Open – File

openProject

Opens the specified project in System Connectivity Manager

Syntax

`openProject <path_to_the_cpm_file>`

where

`path_to_the_cpm_file` Full path to the project (`.cpm`) file to be opened.

Note: If path has spaces, enclose them in quotes.

Examples

Command	Results
`openProject`	Launches the *Open* dialog box. You can now Browse to the project (`.cpm`) file to be opened.
`openProject D:\\design\\modules\\cpu.cpm`	Open the `cpu.cpm` file located in the `D:\design\module` folder **Note:** If backward slash (`\`) is a part of path, specify two slashes (`\\`). This is required because using a single `\` comments out the next alphabet in the path.
`cd D:\\design\\modules` `openProject cpu.cpm`	Same result as `openProject D:\\design\\modules\\cpu.cpm` **Note:** When the `openProject` command is followed by the project filename only the local directory is searched for the specified filename.
`openProject D:/design/modules/cpu.cpm`	Same result as previous example

Menu Text

File – Open Project

packagingOptions

Specifies the options for packaging a block in your design and to alias or mask global signals in the block. This command lets you specify the prefix, suffix, and range options for controlling the value of the reference designators for instances in the block.

Syntax

```
packagingOptions -o/-s <suffix string>/-p <prefix string>/-r [list
<range from> <range to>] -g [list <original global sig> <new global
sig>] -reuse <reuse instance name>
```

where the -o, -s, -p, and -r are mutually exclusive options.

-o	Use optimized packaging. Preserves the reference designators of components in the block
-s <suffix string>	Use suffix. This option lets you and specify the unique suffix to be used for reference designators of components in the block.
-p <prefix string>	Use prefix. This option lets you and specify the unique prefix to be used for reference designators of components in the block.
-r [list <range from> <range to>]	Use Ref. Des. name. This option lets you specify the range of reference designators to be used for components in the block or to modify the reference designator range that was specified when the block was created.
-g [list <original global signal> <new global signal>]	Globals. Optional parameter. Use this option if you want to alias a global signal in the block to a signal in the design in which you are adding the block.

-reuse <reuse instance name>	Physically reuse block. Use this option to specify the unique ID for the instance of the reuse block in your design. Allegro PCB Editor uses this ID to differentiate between multiple instances of a reuse module.

Examples

Command	Results
`packagingOptions`	Displays the Block Packaging Options dialog box.
`packagingOptions -o`	Uses Optimized packaging for placed block.
`packagingOptions -p CNTR`	The reference designators of components in the CONTROLLLER block will be CNTR_U1, CNTR_U2, and so on.

paste

Pastes the contents of the clipboard at the specified or current cursor location.

Syntax

```
paste
```

Syntax to be used in Component Connectivity Details pane

```
paste [special|sp] -u|-l -p|-s value -r findstring replacestring -cb -b -ncol
     <column name>
```

Syntax to be used in Component List pane

```
paste [special|sp] -c|-con|Connectivity -p|-pro|Properties -Comments -n|-num|-
     number <numberofcopies>
```

Syntax to be used in Signal List pane

```
paste [special|sp] -p|-pro|Properties -m|-com|-comments
```

where

special | sp	optional parameter keyword used when you want to specify different options for pasting the copied content
-u	Change the text in the clipboard to uppercase before pasting
-l	Change the text in the clipboard to lowercase before pasting
-p	Add a prefix to the text before pasting
-s	Add a suffix to the text before pasting
value	Prefix or the suffix string
-r	Before pasting, replace the text — specified by `findstring` — in the clipboard with the text — specified by `replacestring`.
findstring	The string to be replaced
replacestring	The string to be used instead of the replaced text
-cb	Pastes the copied content on the clipboard

-b	Ignores the blanks rows while pasting
-c|-con|Connectivity	Pastes the connectivity of the component instance
-p|-pro|Properties	Use this switch to paste the component properties along with the component instance
-Comments	Indicates that the comments on a component are to be pasted along with the component instance
-n|-num|-number	Specifies that the next argument indicates the number of component instances to be pasted
<numberofcopies>	Specifies the number of instances of a component to be pasted
-ncol <column name>	specifies the column number where the action is to be performed

Examples

Example 1: A section the script that copies the termination from one component pin and pastes it to another pin of the same component.

```
...
copy pin [list add14 57] termination
selectObject pin add15
paste
...
...
```

Example2: A section the script that copies the component pin names and process it before pasting them as signal names.

```
copy pin [list i7 dq<1> 7 ] [list i7 dq<2> 10 ] [list i7 dq<3> 23 ] [list i7 dq<4> 26 ]
selectObject pin [list i7 dq<1> 7]
paste sp -u -p left_
...
```

Example 3: Set of commands to copy a component instance and paste it along with the connectivity data

```
copy inst i4
paste -Connectivity -Properties -Comments -Number 1
```

Menu Text

Edit – Paste

Edit – Paste Special

See Also

copy

cut

physicalView

Opens the design in the physical mode. This command is valid only when the design is currently open in the logical mode.

Syntax

```
physicalView -nosave
```

Menu Text

View – Physical View

See Also

logicalView

recomputeDiffPair

Recomputes Differential Pairs differential pairs based on the latest setup options. Based on the modifications made in the setup options, this may result in the creation or deletion of new differential pairs.

Syntax

```
recomputeDiffPair <inst name>
```

Menu Text

Object — Recompute Differential Pairs

redo

Repeats the last action.

Syntax

```
redo
```

Menu Text

Edit – Redo

See Also

undo

regenPhysicalNetNames

Used for Regenerating the Physicals Net Names.

Syntax

```
regenPhysicalNetNames
```

Menu Text

Project – Regenerate Physical NetName

reImportBlock

Re-imports a read-only block.

Syntax

```
reImportBlock <library name> <cell name>
```

where

`<library name>`	The library where the block is stored.
`<cell name>`	The cell which represents the block.

Examples

Command	Result
`reImportBlock hier_design_lib clock`	Reimports the block named `clock` from the library `hier_design_lib`.

Menu Text

Select a block and choose *Re-import Block* from the context menu.

Menu Text

Object – Edit Co-design – Die

reimportVerilog

Reimports the specified Verilog file into design database

Syntax

```
reimportVerilog <path of verilog file> <lib> [-useInstanceNameAsRefdes] [-
     importInterfacesOnly <module name>] [-m <map file name>]
```

where

path	Path to the Verilog file to be imported
lib	Design library in which the block created by importing the verilog file is saved
`-useInstanceNameAsRefdes`	Optional When specified, this argument indicates that for all the components in the Verilog file, instance names will be used as reference designators
`-importInterfacesOnly`	Optional When specified, indicates that only interface data is to be imported for the module
`<module name>`	Optional The name of the module for which the interface data is to be imported
`-m`	Optional When specified, indicates that a map file is used to map the component names in the Verilog file to component names in the libraries added for the current project.
`map <filename>`	Optional The name of the map file used for mapping

Example

Command	Result
`importVerliog`	Launches the Import Verilog dialog box
`importVerilog D:\\design_samples/ user_pins_added.v tutorial_lib -r -t`	All the modules in the Verilog file, `user_pins_added.v`, are imported to the `tutorial_lib` library as table blocks
`importVerilog D://verilog_import/ top.v tutorial_lib -importInterfacesOnly adc_top`	The interface data for the `adc_top` module in the `top.v` Verilog file is imported as a table block in the `tutorial_lib` file.

Menu Text

Project — Import — Verilog

removeDiffPairPin

Removes the specified Differential Pair pin.

Syntax

```
removeDiffPairPin[list <instance name> <pin name> <pin number>]
```

where

`<instance name>`	The instance name of the component.
`<Pin name>`	The name of the differentadifferentiall pair pin to remove.
`<Pin number>`	The number of the differential pair pin to remove.

Example

Command	Result
`removeDiffPairPin [list i3 tdo 73]`	Removes the pin number 73.

removeModel

Removes the device model assigned to a component instance or pin. This command works when either the Component List pane or the Component Connectivity Details pane is the active window.

Syntax

```
removeModel inst|pin instance name|[list instance_name pinname pin_number]
```

where

`inst`	keyword used to remove device model from a component instance
`pin`	keyword used to remove device model from component pin
`instance name`	Name of the instance from which the model is to be removed **Note:** Specified when `inst` is used as the second argument
`[list instance_name pinname pin_number]`	Details specified to identify the component pin from which the model is to be removed

Examples

Command	Result
`removeModel inst`	Removes device model assigned to the component instance selected in the Component List pane.
`removeModel pin [list i12 dout3 10 ]`	Removes device model assigned to pin 10, `dout3`, of component instance `i12`

Menu Text

Object – SI Model – Remove Model

See Also

assignModel

removeTool

Removes any user or TCL/TK tools that you have added to System Connectivity Manager.

Syntax

```
removeTool [user/tcl] <menu text>
```

where

`user/tcl`	Keyrword to define the type of the tool: user or TCL/TK. Either use user or tcl.
`<menu text>`	The menu text associated with the tool that needs to be removed.

Examples

Command	Result
`removeTool user PDV`	Removes PDV from the Tools menu.
`removeTool user "Parts Developer"`	Removes Parts Developer from the Tools menu.

renameConnection

Use this command to rename the connection name.

Syntax

```
renameConnection [-RC] [list <instance name> <pin name> <pin number> <new signa
     name>]
```

where

`[-RC]`	This option enables you to retain the existing constraints.
`<instance name>`	Instance of the component.
`<pin name>`	The pin name
`<pin number>`	The pin number
`<new signal name>`	The new name for the signal

Example

Command	Results
`renameConnection [list i2 clock 2 CLK]`	Renames the signal connected to pin 2 of instance i2 to CLK.

renameObject

Changes the name of the specified or the selected component instances or signals.

Syntax

```
rename inst | net <new name>

rename inst | net [list <old name> <new name>]|[list dp <diffpair name> <new
     diffpair name>]
```

where

inst	keyword used to when a component instance is to be renamed
net	keyword used to specify that nets are being renamed
new name	Name to be assigned to the selected instance or net
old name	Existing instance name or the net name
dp	keyword used when differential pair signals are to be renamed
diffpair name	Existing differential pair name
new diffpair name	New name for the differential pair name

Example

Command	Results
`rename inst j55`	Changes the instance name of the component currently selected in the Component List pane to `j55`
`rename inst [list i2 c_i2] [list i3 c_i3]`	Component instances `i2` and `i3` are renamed to `c_i2` and `c_i3`, respectively
`rename net [list woit wait] [list dp cvx cyprus]`	Signal `woit` is renamed to `wait`, and the differential pair signal `cvx` is renamed as `cyprus`.

Menu Text

Object – Change – Name

replaceComp

Replaces the specified component instance with the component from a specified library.

Note: Depending of the setup options specified in the Component REplace tab of the Setup dialog box, the connectivity and property information on the component may or may not be preserved.

Syntax

```
replaceComp <name of instance> -l newlib -c newcell -v newview -k [list propname
     propvalue] -i [list propname propvalue] -m pinname|pinnumber|both -x [list
     command args]
```

where

`name of instance`	keyword used when a component instance is to be renamed
`-l`	keyword used to specify that nets are being renamed
`newlib`	Name to be assigned to the selected instance or net
`-c`	Existing instance name or the net name
`newcell`	keyword used when differential pair signals are to be renamed
`-v newview`	keyword used for existing differential pair name
`-k [list propname propvalue]`	Key property values assigned to the new components
`-i [list propname propvalue]`	Injected property values assigned to the new components
`-m pinname|pinnumber|both`	Specifies whether the mapping between the old component and the new component is to be done using pin names, pin numbers, or by using both.

`-x [list command args]`

Specifies the user mapping.

.The valid values for command are:

- AddSignal
- AddProperty

If the value of the parameter after the list keyword is `AddSignal`, the valid values for args are:

- sourcePinName
- sourcePinNumber
- targetPinName
- targetPinNumber

For `AddProperty`, the valid values of args are:

- PropertyName
- PropertyValue

Example

Command	Results
`replaceComp i1 -l classlib -c tc55b4257 -v chips -k [list PACK_TYPE SOIC ] [list PART_NAME TC55B4257 ] -m both`	Component instance `i1` is replaced with the component `tc55b4257` from the `classlib` library.
`replaceComp i2 -l classlib -c tc55b4257 -v sym_1 -k [list PACK_TYPE SOIC ] [list PART_NAME TC55B4257 ] -m pinname`	Component instance `i2` is replaced with the schematic symbol of component `tc55b4257` from the `classlib` library. Pin names are used to map the pins of old component to the pins of the new component.

Command	Results
`replaceComp i2 -l classlib -c tc55b4257 -v sym_1 -k [list PACK_TYPE SOIC] [list PART_NAME TC55B4257] -m pinname -x [list AddSignal a 1 newa 3] [list AddSignal b 2 newb 4] [list AddProperty ABC XYZ]`	Component instance `i2` is replaced with the schematic symbol of component `tc55b4257` from the `classlib` library.

Menu Text

Object – Replace Component

runDRC

Runs the design rule checks on the design currently open in System Connectivity Manager.

Syntax

```
runDRC
```

Menu Text

Project – Design Rules Check

resolveViolation

Resolves the specified violation listed in the Visual Design Differences window.

Syntax

```
resolveViolation -number <row number of violation>
```

where

`-number <row number of violation>`	Specifies the row number of the violation in the Visual Design Differences window.

Example

Command	Result
`resolveViolation -number 3`	Resolves the violation listed in the third row of the Visual Design Differences window.

saveAll

Saves all the files in the design

Syntax

```
saveAll
```

See Also

saveDesign

saveDesign

Saves the design

Syntax

```
saveDesign
```

See Also

saveDesign

saveDesignAs

Makes the copy of the current design and saves it as a block in the specified library.

Syntax

```
saveDesignAs <new library name> <new block name>
```

where

new library name	Name of the design library in which the design block is to be saved
new block name	Name of the block

Example

Command	Result
`saveDesignAs`	Launches the Block Save As dialog box
`saveDesignAs tutorial_lib adder`	Saves a copy of the design currently open in the master mode, as `adder` block in the `tutorial_lib` library.

Menu Text

File – Save As

saveSignal

Exports the selected signals to the specified signal (`.sig`) file.

Syntax

```
saveSignal filepath
```

where

filepath	Path to the signal (`.sig`) file in which the selected signals are to be saved.

Example

Command	Result
`saveSignal`	Launches the Save File As dialog box
`saveSignal d:/siglist.sig`	Saves the selected signals to the `siglist.sig` file.
`selectObject net rcs0 rcs1 rcs2` `saveSignal d:/siglist.sig`	Saves the signals `rcs0`, `rcs1`, and `rcs2` to the `siglist.sig` file.

Menu Text

Design – Save Signals To File

See Also

loadSignal

selectAssocComp

Selects a row in the associated component window using the row's reference designator.

Syntax

```
selectAssocComp <reference designator>
```

where

`reference designator`	Indicates the name of the row selected using the reference designator.

Example

Command	Result
`selectAssocComp R6`	R6 represents the selected ref des row.

selectAssocCompNet

Selects a net in the associated component window.

Syntax

```
selectAssocCompNet [list <reference designator> <net name>]
```

where

`list`	Indicates the refdes and net pairs that identify a row in the assoc comp window.

Example

Command	Result
`selectAssocCompNet [list R6 Net_series_1]`	`R6 Net_series_1` indicates the refdes and net pairs that identify a row in the assoc comp window.

selectAssocCompParent

Selects the parent column of the given associated component.

Syntax

```
selectAssocCompParent [list <reference designator> <parent instance name>]
```

where

`file`	Indicates the refdes parent inst value pairs to identify a row in the assoc comp window.

Example

Command	Result
`selectAssocCompParent [list R6 i1]`	R6 indicates the refdes parent inst value pairs to identify a row in the assoc comp window.

selectAssocCompParentPin

Selects the parent pin of the given associated component.

Syntax

```
selectAssocCompParentPin [list <reference designator> <parent instance name> <pin name>]
```

where

`list`	Indicates the pair of refdes, parent instance name, and parent instance pin name.

Example

Command	Result
`selectAssocCompParentPin [list R6 i1 2]`	`R6 i1 2` indicates the pair of refdes, parent instance name, and parent instance pin name.

selectAssocCompPin

Selects the associated component pin in the Assoc Comp Window.

Syntax

```
selectAssocCompPin [list <reference designator> <pin number>]
```

where

`list` Indicates the pair of refdes and pin number.

Example

Command	Result
`selectAssocCompPin [list R6 2]`	`R6 2` indicates the pair of refdes and pin number.

selectObject

Selects the specified design object, such as a component instance, a signal, or a component pin.

Syntax

```
selectObject inst <instance name>
selectObject net <signal name>
selectObject pin [list <instance name> <pin name> <pin number>]
```

where

`inst|net|pin`	Keyword specifying the object type to be selected. ■ `inst` - used for selecting a component instance in the Component List pane ■ `net` - used for selecting a signal in the Signal List pane ■ `pin` - used for selecting a component pin
`<instance name>`	Name of the instance to be selected
`<signal name>`	Name of the signal or the net to be selected
`<pin name>`	Name of the pin to be selected.
`<pin number>`	Pin number of the pin to be selected using the selectObject command

Example

Command	Result
`selectObject inst i1 i2 i3`	Selects component instances `i1`, `i2`, and `i3` in the Component List pane
`selectObject inst all`	Selects all component instances in the design
`selectObject net a b[0]`	Selects nets `a` and `b[0]` in the Signal List pane

Command	Result
`selectObject pin [list i1 data[0] 9]`	Selects pin `9` of instance `i1`.
`selectObject net [list dp DP_add]`	Selects differential pair signal `DP_add`.

selectWindow

Makes the specified window in the System Connectivity Manager interface as the active window

Syntax

```
selectWindow <window name>
```

where

window_name — The window or the pane in System Connectivity Manager that is to be made the active window. The valid values are:

- clp – indicates Component List pane
- slp – indicates Signal List pane
- ccp – indicates Component Connectivity Details pane
- clipboard – indicates Clipboard
- hierarchy – activates Hierarchy Viewer
- fileviewer– activates File Viewer
- signalnavigate – activates Signal Navigate
- propwindow – Properties Window
- assoccompview – Assoc Compon Viewer
- sessionlog – Make the Session log as active vi
- violations – Activates the Violations tab
- designdiffview – Activates the Visual Design Differences tab in the Violations window

Note: To view the commands for making Signal Connectivity Details pane as the active window, see Example 3 on page 172.

Examples

Command	Result
`selectWindow clp`	Makes the Component List pane the active window

Example 1

When `DSMAIN-327` error message is thrown. The message states:

> *The System Connectivity Manager pane that needs to be active for this command to work is not active. Select the appropriate pane before running the command. Use help <command name> for more details.*

a. Launch System Connectivity Manager.

b. Open the TCL shell by selecting *File – Open TCL Shell*.

c. In the TCL shell type `addSignal` and press Enter.

If you receive `DSMAIN-327` message, it indicates that currently any other pane, besides the Signal List pane, is the active window. To resolve this error, type following commands.

```
selectWindow slp
addSignal
```

Example 2

In a TCL script, to change the focus area. A section of the TCL script that uses the `selectWindow` command is shown.

```
...
selectWindow slp
changeVoltage dout 5
selectWindow clp
changeRefdes [list i1 U5 ]
...
```

Example 3

- TCL commands for making Signal Connectivity Details pane as the active window

```
selectObject net brd
editConnectivity
selectWindow ccp
```

See Also

selectObject

setOptions

Use this command to specify the set up options.

Syntax

```
setOptions <category name> <option> <values>
setOptions -help
setOptions <category> -help
```

where

`<category>`	Use this argument to specify the category
`<option>`	Use this argument to specify the option.
`<value>`	Use this argument to specify the value to be set.

The following table lists all the categories and the relevant options.

Category	Option	Value	Notes
library	add	<library name>/all	library name must be from the list of available libs
	remove	<library name>/all	
	design_library	<library name>	
	design_cell	<cell name>	
drc	<drc function name> <error/ info/warning/off>]		
general	VectorSqrBracket	on/off	
	remove_violations_when_fixed	on/off	
	export_local_as_inout	on/off	

Category	Option	Value	Notes
verification	onload		
	timed	10min/1hr	
	manual		
diffpair	dppin_rules	[list <negative pin string> prefix/ suffix <positive pin string> prefix/ suffix]	list of set of 4 values
	dpsig_rules	[list <negative> <positive> suffix/ prefix]	
	dppin_prefix	<prefix string>	
	dpsig_prefix	<prefix string>	
	auto_connect_dp leg	on/off	
packager	reuse_refdes	on/off	
	default_phys_de s_prefix (this is refdes prefix)	<string>	
	ref_des_length	<numeral>	
	part_type_length	<numeral>	
	net_name_length	<numeral>	
	sd_suffix_separa tor	<string>	
	sd_prefix_separa tor	<string>	
	net_name_chars	[list 0-9 a-z @]	

Category	Option	Value	Notes
ppt	add	<path to ppt file>	
	exclude	<ppt to exclude>	
	include	<ppt to include>	
	use_cell_level		
	merge		
	case_sensitive_row_match		
verilognetlist	ignorebypasscap		
	ignorepupd		
	ignoreterm		
	single_file_nlist		
	vlog_uppercase		
	map_by_position		
	bind_design (regen configuration)		
	ignoreCompProperty	<property name>	
	default_net	<wire>	
	max_hdldirect_errors	<number>	
	timescale	<time scale eg 1ns/1ns	
	supply	[list signal_name <supply 0> <supply 1>]	supply values are 0 or 1

Example

Command	Result
`setOptions diffpair -` `auto_connect_dpleg off`	Sets options auto_connect differential pair leg to off.

Menu Text

Project – Settings

setViewOption

Sets the view options in the Signal list pane or Component Connectivity Details pane. You can set how to display whether to show expanded differential pairs and vector signals.

To run this command you must have focus on the Component Connectivity Details pane or the Signal List pane.

Syntax

```
setViewOptions pin/net bus_on/bus_off dp_on/dp_off
```

where

`pin/net`	Pin: Applies the settings to the Component Connectivity Details pane. Net: Applies the settings to the Signal List pane.
`bus_on/bus_off`	If on displays buses instead of individual pins.
`dp_on/dp_off`	If on displays differential pair pins instead of individual pins.

Example

Command	Result
`setViewOptions pin bus_off dp_off`	Sets options for Component Connectivity pane to display buses and differential pins as individual pins.
`setViewOptions pin bus_on dp_on`	Sets options for Component Connectivity pane to display buses and differential pairs.
`setViewOptions net bus_off dp_off`	Sets options for Signals List pane to display buses and differential pins as individual pins.

siSetup

Modifies the configuration of the device model library (`.dml`) for your project. Use this command to add new device model libraries to the active project or to remove existing device model libraries. Command can also be used to specify the working library for the project.

Syntax

```
siSetup add|delete|setworking <filename1 filename2 filename3 ...>
```

where

`add`	Keyword used to include a new device model library to the project
`delete`	Keyword used to remove a device model library from the project
`setworking`	Keyword used to specify a device model library as the working library for the project
`filename`	Path to the .dml or .ndx files to be added to the project

Example

Command	Result
`siSetup add D:/design_samples/ si_model/tutorial.dml`	Includes the `tutorial.dml` located in the `D:/design_samples/si_model` folder to the project
`siSetup delete D:\\si_model\\trans.dml`	Removes the `trans.dml` from the project
`siSetup setworking D:\\si_model\\trans.dml`	Makes the specified device model library as the working library for the project. All

sort

Sorts the rows in the specified pane based on the specified column. If rows are unsorted, this command sorts in ascending order, and using the command again sorts in descending order.

Syntax

```
sort inst/pin/net <columnname>
```

where

`inst/pin/net`	Keyword to indicate the pane in which perform a sort: inst: Component List pane. pin: Component Connectivity details pane. net: Signals List pane.
`<columnname>`	The name of the column using which to perform the sort. Incase <columnname> is not specified, the default sort is performed using instance, pin, and net.

Example

Command	Result
`sort inst`	Sorts the components in the Components List by Instance.
`sort inst "Ref Des"`	Sorts the components in the Components List by Ref Des.
`sort pin "pin number"`	Sorts the Pins in the Components List by Pin number.

swapSignals

Used to swap t

Syntax

```
swapSignal [list <instance name> <pin name1> <pin number1>] [list <instance
name> <pin name2> <pin number2>]
```

where

[list <instance name> <pin name1> <pin number1>]	Instance, pin name, and pin number to specify the first signal.
[list <instance name> <pin name2> <pin number2>]	Instance, pin name, and pin number to specify the second signal.

Example

Command	Result
`swapSignal [list i1 1 1] [list i1 2 2]`	Swaps the signals connected to pins 1 and 2 of the component i1.

switchTab

Changes focus to an open tab in the design.

Syntax

```
switchTab <design name> [design path]
```

where

`<design name>`	The name of the tab in which the design is open.
[design path]	Optional parameter to specify the path to the design, in case multiple tabs with the same name are open in context. For example, if you have two instances of a block named "mid" inside the "top" names i1 and i2 and both are open.

Example

Command	Result
`switchTab processor`	Switches focus from the currently selected tab to the tab named "processor".
`switchTab root`	Switches focus from the currently selected tab to the tab named "root".
switchTab mid top.i1	Switches focus from the currently selected tab to the tab named "mid" located at the path top.i1.

trackInSignalList

Highlights the specified signal or the signal selected in the Component Connectivity Details pane in the Signals List pane.

Syntax

```
trackInSignalList [list <instance name> <pin name> <pin number>
```

where

`<instance name>`	The instance to which the signal is attached.
<pin name>	The pin name of the pin to which the signal is attached.
<pin number>	The pin number of the pin to which the signal is attached.

Example

Command	Result
`trackInSignalList`	Selects the signal currently selected in Components Connectivity Details pane in the Signals List pane.
`trackInSignalList [list i2 q1<0> 10]`	Selects the signal connected to pin number 10 of component instance i2 in the Signals List pane.

Menu Text

Object – Track Signal in Signal List

unAliasSignal

Removes the alias created between the two specified signal. The names of the signals to be unaliased is passed as parameters to this command.

Syntax

```
unAliasSignal [list <first_net_name> <lsb> <msb>] [list <second_net_name> <lsb>
     <msb>]
```

where

`first_net_name`	names of the signals to be aliased
`second_net_name`	
`lsb`	Least significant bit
`msb`	Most significant bit

Examples

Command	Result
`unAliasSignal aa bb`	Removes the alias created between net `aa` and net `bb`
`unAliasSignal [list a 4 8] [list b 0 4]`	Removes alias between vector signals `a<8..4>` and `b<4..0>`

Menu Text

Object – Remove Alias

See Also

aliasSignal

undo

Reverses the last action.

Syntax

```
undo
```

Menu Text

Edit – Undo

See Also

redo

updateBlockSource

Used to update the block source.

Syntax

```
updateBlockSource -lib <library name> -cell <cell name> -cds /-cpm <file path>
```

where

`-lib <library name>`	Specifies the library name of the block.
`-cell <cell name>`	Specifies the cell name of the block.
`-cds/-cpm`	Keyword to indicate whether the file specified is cds or cpm file.
`<file path>`	Specifies the path to the cds or cpm file.

updateDifference

Updates differences displayed in the Visual Design Differences window. You must select the Visual Design Differences tab for this command to work successfully.

Syntax

```
updateDifference -all -silent
```

Or

```
updateDifference [list <category> <object> <app1 value> <app2 value>] -silent
```

where

`-all`	Updates all differences
`-silent`	Optional parameter, to suppress Merging Design Differences dialog box.
`<category>`	The category of the difference to be updated.
`<object>`	The object to be updated.
`<app1 value>`	The existing value in the design.
`<app2 value>`	The new value, to be updated from the physical file.

Examples

Command	Result
`updateDifference -all -silent`	Updates all differences reported in VDD, and suppresses the Merging Design Differences dialog box.
`updateDifference [list "Component Property Differences" "C1#1 (capacitor)" CURRENT=CIMAX " " ] [list "Component Property Differences" "C1#1 (capacitor)" DIST=FLAT " " ]`	Updates two Component Property Differences, and removes the existing properties.

Menu Text

Visual Design Differences – Update

updateImportedBlock

Use this command to update any imported blocks in your design.

Syntax

```
updateImportedBlock
```

Menu Text

Project – Update Imported Blocks

updateVersion

Updates the version of the specified block or component in the design.

Syntax

```
updateVersion [-design <design name>] -lib <library name> -cell <cell name> [-view
     <view name>] [-option none/both/pinname/pinnumber]
```

where

`[ -design <design name>`	Optional parameter to specify the design. If not provided, the current design is considered.
`-lib <library name>`	The name of the library where the block or component is located.
`-cell <cell name>`	The name of the cell where the block or component is located.
`[-view <view name>`	Optional parameter to specify the view to be updated.If omitted, all views are updated.
`[-option none/both/ pinname/ pinnumber]`	Incase the component has different sets of pins after update, use these options to specify the mapping information: ■ none: do not map. ■ pinname: map using pin name. ■ pinnumber: map using pin number. ■ both: map using both pin name and pin number.

Examples

Command	Result
`updateVersion -design root -lib standard -cell R -view Sym_1 -option both`	Updates the versions of both the pin name and pin number of the component standard:R:Sym_1 placed in the root design.

validateRevisions

Lets you update all the instances of the component used in the design with the latest version of the component in the component library.

Syntax

```
validateRevisions
```

Menu Text

Project — Validate Revisions.