Tcl Commands Reference Guide

PolarFire FPGA

NOTE: PDF files are intended to be viewed on the printed page; links and cross-references in this PDF file may point to external files and generate an error when clicked. **View the online help included with software to enable all linked content.**

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email:
sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at **www.microsemi.com.**

5-02-00754-5/03.18

Table of Contents

Introduction to Tcl Scripting

Tcl, the Tool Command Language, pronounced *tickle*, is an easy-to-learn scripting language that is compatible with Libero SoC software. You can run scripts from either the Windows or Linux command line or store and run a series of commands in a *.tcl batch file.

This section provides a quick overview of the main features of Tcl:

- Basic syntax
- Types of Tcl commands
- Variables
- Command substitution
- Quotes and braces
- Lists and arrays
- Control structures
- Print statement and Return values

For complete information on Tcl scripting, refer to one of the books available on this subject. You can also find information about Tcl at web sites such as http://www.tcl.tk.

Libero SoC provides additional capabilities and built-in Tcl Commands:

- Running Tcl scripts from the command line
- Exporting Tcl scripts
- extended_run_lib
- Tcl Commands as specified in this document

Tcl Command Documentation Conventions

The following table shows the typographical conventions used for the Tcl command syntax.

Syntax Notation	Description
`command -argument`	Commands and arguments appear in Courier New typeface.
`variable`	*`Variables appear in blue, italic Courier New typeface. You must substitute an appropriate value for the variable.`*
[-argument*value*] [*variable*]+	Optional arguments begin and end with a square bracket with one exception: if the square bracket is followed by a plus sign (+), then users must specify at least one argument. The plus sign (+) indicates that items within the square brackets can be repeated. Do not enter the plus sign character.

Note: All Tcl commands are case sensitive. However, their arguments are not.

Examples

Syntax for the get_clocks command followed by a sample command:

```
get_clocks variable
get_clocks clk1
```

Syntax for the backannotate command followed by a sample command:

```
backannotate -name file_name -format format_type -language language -dir directory_name [-
netlist] [-pin]
```

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

Wildcard Characters

You can use the following wildcard characters in names used in Tcl commands:

Wildcard	What it Does
\	Interprets the next character literally
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Note: The matching function requires that you add a slash (\) before each slash in the port, instance, or net name when using wildcards in a PDC command. For example, if you have an instance named “A/B12” in the netlist, and you enter that name as “A\\/B*” in a PDC command, you will not be able to find it. In this case, you must specify the name as A\\\\/B*.

Special Characters [ ], { }, and \

Sometimes square brackets ([ ]) are part of the command syntax. In these cases, you must either enclose the open and closed square brackets characters with curly brackets ({ }) or precede the open and closed square brackets ([ ]) characters with a backslash (\). If you do not, you will get an error message.

For example:

```
pin_assign -port {LFSR_OUT[0]} -pin 15
or
pin_assign -port LFSR_OUT\[0\] -pin 180
```

Note: Tcl commands are case sensitive. However, their arguments are not.

Entering Arguments on Separate Lines

To enter an argument on a separate line, you must enter a backslash (\) character at the end of the preceding line of the command as shown in the following example:

```
backannotate -dir \
{..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

See Also

Introduction to Tcl scripting

Basic syntax

Basic Syntax

Tcl scripts contain one or more commands separated by either new lines or semicolons. A Tcl command consists of the name of the command followed by one or more arguments. The format of a Tcl command is:

```
command arg1 ... argN
```

The command in the following example computes the sum of 2 plus 2 and returns the result, 4.

```
expr 2 + 2
```

The **expr** command handles its arguments as an arithmetic expression, computing and returning the result as a string. All Tcl commands return results. If a command has no result to return, it returns an empty string.

To continue a command on another line, enter a backslash (\) character at the end of the line. For example, the following Tcl command appears on two lines:

```
import -format "edif" -netlist_naming "Generic" -edif_flavor "GENERIC" {prepi.edn}
```

Comments must be preceded by a hash character (#). The comment delimiter (#) must be the first character on a line or the first character following a semicolon, which also indicates the start of a new line. To create a multi-line comment, you must put a hash character (#) at the beginning of each line.

Note: Be sure that the previous line does not end with a continuation character (\). Otherwise, the comment line following it will be ignored.

Special Characters

Square brackets ([ ]) are special characters in Tcl. To use square brackets in names such as port names, you must either enclose the entire port name in curly braces, for example, pin_assign -port {LFSR_OUT[15]} -iostd lvttl -slew High, or lead the square brackets with a slash (\) character as shown in the following example:

```
 pin_assign -port LFSR_OUT\[15\] -iostd lvttl -slew High
```

Sample Tcl Script

```
#Create a new project and set up a new design

new_project -location {D:/2Work/my_pf_proj} -name {my_pf_proj} -project_description {}\
-block_mode 0 -standalone_peripheral_initialization 0 -use_enhanced_constraint_flow 1\
-hdl {VERILOG} -family {PolarFire} -die {MPF300TS_ES} -package {FCG1152} -speed {-1} \
-die_voltage {1.0} -part_range {EXT} -adv_options {IO_DEFT_STD:LVCMOS 1.8V}\
-adv_options {RESTRICTPROBEPINS:1} -adv_options {RESTRICTSPIPINS:0}\
-adv_options {SYSTEM_CONTROLLER_SUSPEND_MODE:1} -adv_options {TEMPR:EXT} \
-adv_options {VCCI_1.2_VOLTR:EXT} -adv_options {VCCI_1.5_VOLTR:EXT} \
-adv_options {VCCI_1.8_VOLTR:EXT} -adv_options {VCCI_2.5_VOLTR:EXT} \
-adv_options {VCCI_3.3_VOLTR:EXT} -adv_options {VOLTR:EXT}

#Import HDL source file

import_files -convert_EDN_to_HDL 0 -hdl_source {C:/test/prep1.v}

#Import HDL stimulus file

import_files -convert_EDN_to_HDL 0 -stimulus {C:/test/prep1tb.v}

#set the top level design name

set_root -module {prep1::work}

#Associate SDC constraint file to Place and Route tool

organize_tool_files -tool {PLACEROUTE} -file {D:/2Work/my_pf_proj/constraint/user.sdc} \
-module {prep1::work} -input_type {constraint}

#Associate SDC constraint file to Verify Timing tool

organize_tool_files -tool {VERIFYTIMING} -file
{D:/2Work/my_pf_proj/constraint/user.sdc}\
 -module {prep1::work} -input_type {constraint}

#Run synthesize

run_tool -name {SYNTHESIZE}

#Configure Place and Route tool

configure_tool -name {PLACEROUTE} -params {DELAY_ANALYSIS:MAX} -params
{EFFORT_LEVEL:false}\
 -params {INCRPLACEANDROUTE:false} -params {MULTI_PASS_CRITERIA:VIOLATIONS}\
 -params {MULTI_PASS_LAYOUT:false} -params {NUM_MULTI_PASSES:5} -params {PDPR:false}\
 -params {RANDOM_SEED:0} -params {REPAIR_MIN_DELAY:false} -params
{SLACK_CRITERIA:WORST_SLACK} \
 -params {SPECIFIC_CLOCK:} -params {START_SEED_INDEX:1} -params
{STOP_ON_FIRST_PASS:false}\
 -params {TDPR:true}
#Run Place and Route
run_tool -name {PLACEROUTE}
#Configure Timing Report Generation
configure_tool -name {VERIFYTIMING} -run_tool -name {PLACEROUTE}params
{CONSTRAINTS_COVERAGE:1}\
-params {FORMAT:XML} -params {MAX_TIMING_FAST_HV_LT:0} -params {MAX_TIMING_SLOW_LV_HT:1}
\
-params {MAX_TIMING_SLOW_LV_LT:0} -params {MAX_TIMING_VIOLATIONS_FAST_HV_LT:0} \
-params {MAX_TIMING_VIOLATIONS_SLOW_LV_HT:1} -params
{MAX_TIMING_VIOLATIONS_SLOW_LV_LT:0}\
-params {MIN_TIMING_FAST_HV_LT:1} -params {MIN_TIMING_SLOW_LV_HT:0} -params
{MIN_TIMING_SLOW_LV_LT:0} -params {MIN_TIMING_VIOLATIONS_FAST_HV_LT:1} -params
{MIN_TIMING_VIOLATIONS_SLOW_LV_HT:0} \
-params {MIN_TIMING_VIOLATIONS_SLOW_LV_LT:0}


 #Run Verify Timing tool
run_tool -name {VERIFYTIMING}
#Run Power Verification tool
run_tool -name {VERIFYPOWER}
#Export bitstream
export_bitstream_file -file_name {prep1} \
-export_dir {D:\2Work\my_pf_proj\designer\prep1\export} -format {STP} -master_file 0 \
-master_file_components {} -encrypted_uek1_file 0 -encrypted_uek1_file_components {} \
-encrypted_uek2_file 0 -encrypted_uek2_file_components {} \
-trusted_facility_file 1 -trusted_facility_file_components {FABRIC}
```

Types of Tcl commands

This section describes the following types of Tcl commands:

- Built-in commands
- Procedures created with the proc command

Built-in commands

Built-in commands are provided by the Tcl interpreter. They are available in all Tcl applications. Here are some examples of built-in Tcl commands:

- Tcl provides several commands for manipulating file names, reading and writing file attributes, copying files, deleting files, creating directories, and so on.
- exec - run an external program. Its return value is the output (on stdout) from the program, for example:

```
set tmp [ exec myprog ]
puts stdout $tmp
```

- You can easily create collections of values (lists) and manipulate them in a variety of ways.
- You can create arrays - structured values consisting of name-value pairs with arbitrary string values for the names and values.
- You can manipulate the time and date variables.

- You can write scripts that can wait for certain events to occur, such as an elapsed time or the availability of input data on a network socket.

Procedures created with the proc command

You use the proc command to declare a procedure. You can then use the name of the procedure as a Tcl command.

The following sample script consists of a single command named **proc**. The proc command takes three arguments:

- The name of a procedure (myproc)
- A list of argument names (arg1 arg2)
- The body of the procedure, which is a Tcl script

```
proc myproc { arg1 arg2 } {
# procedure body
}
myproc a b
```

Variables

With Tcl scripting, you can store a value in a variable for later use. You use the set command to assign variables. For example, the following set command creates a variable named x and sets its initial value to 10.

```
set x 10
```

A variable can be a letter, a digit, an underscore, or any combination of letters, digits, and underscore characters. All variable values are stored as strings.

In the Tcl language, you do not declare variables or their types. Any variable can hold any value. Use the dollar sign ($) to obtain the value of a variable, for example:

```
set a 1
set b $a
set cmd expr
set x 11
$cmd $x*$x
```

The dollar sign $ tells Tcl to handle the letters and digits following it as a variable name and to substitute the variable name with its value.

Global Variables

Variables can be declared global in scope using the Tcl global command. All procedures, including the declaration can access and modify global variables, for example:

```
global myvar
```

Command substitution

By using square brackets ([]), you can substitute the result of one command as an argument to a subsequent command, as shown in the following example:

```
set a 12
set b [expr $a*4]
```

Tcl handles everything between square brackets as a nested Tcl command. Tcl evaluates the nested command and substitutes its result in place of the bracketed text. In the example above, the argument that appears in square brackets in the second set command is equal to 48 (that is, 12* 4 = 48).

Conceptually,

```
set b [expr $a * 4]
```

expands to

```
set b [expr 12 * 4 ]
```

and then to

```
set b 48"
```

Quotes and braces

The distinction between braces ({ }) and quotes (" ") is significant when the list contains references to variables. When references are enclosed in quotes, they are substituted with values. However, when references are enclosed in braces, they are not substituted with values.

Example

With Braces	With Double Quotes
set b 2	set b 2
set t { 1 $b 3 }	set t " 1 $b 3 "
set s { [expr $b + $b] }	set s " [expr $b + $b] "
puts stdout $t	puts stdout $t
puts stdout $s	puts stdout $s

will output

```
1 $b 3                         vs.           1 2 3
[ expr $b + $b ]                            4
```

Filenames

In Tcl syntax, filenames should be enclosed in braces { } to avoid backslash substitution and white space separation. Backslashes are used to separate folder names in Windows-based filenames. The problem is that sequences of “\n” or “\t” are interpreted specially. Using the braces disables this special interpretation and specifies that the Tcl interpreter handle the enclosed string literally. Alternatively, double-backslash “\\n” and “\\t” would work as well as forward slash directory separators “/n” and “/t”.For example, to specify a file on your Windows PC at c:\newfiles\thisfile.adb, use one of the following:

```
{C:\newfiles\thisfile.adb}
C:\\newfiles\\thisfile.adb
"C:\\newfiles\\thisfile.adb"
C:/newfiles/thisfile.adb
"C:/newfiles/thisfile.adb"
```

If there is white space in the filename path, you must use either the braces or double-quotes. For example:

```
C:\program data\thisfile.adb
```

should be referenced in Tcl script as

```
{C:\program data\thisfile.adb} or "C:\\program data\\thisfile.adb"
```

If you are using variables, you cannot use braces { } because, by default, the braces turn off all special interpretation, including the dollar sign character. Instead, use either double-backslashes or forward slashes with double quotes. For example:

```
"$design_name.adb"
```

Note: To use a name with special characters such as square brackets [ ], you must put the entire name between curly braces { } or put a slash character \ immediately before each square bracket.

The following example shows a port name enclosed with curly braces:

```
pin_assign -port {LFSR_OUT[15]} -iostd lvttl -slew High
```

The next example shows each square bracket preceded by a slash:

```
pin_assign -port LFSR_OUT\[15\] -iostd lvttl -slew High
```

Lists and arrays

A list is a way to group data and handle the group as a single entity. To define a list, use curly braces { } and double quotes " ". For example, the following set command {1 2 3 }, when followed by the list command, creates a list stored in the variable "a." This list will contain the items "1," "2," and "3."

```
set a { 1 2 3 }
```

Here's another example:

```
set e 2
set f 3
set a [ list b c d [ expr $e + $f ] ]
puts $a
```

displays (or outputs):

```
b c d 5
```

Tcl supports many other list-related commands such as lindex, linsert, llength, lrange, and lappend. For more information, refer to one of the books or web sites available on this subject.

Arrays

An array is another way to group data. Arrays are collections of items stored in variables. Each item has a unique address that you use to access it. You do not need to declare them nor specify their size.

Array elements are handled in the same way as other Tcl variables. You create them with the set command, and you can use the dollar sign ($) for their values.

```
set myarray(0) "Zero"
set myarray(1) "One"
set myarray(2) "Two"
for {set i 0} {$i < 3} {incr i 1} {
```

Output:

```
Zero
One
Two
```

In the example above, an array called "myarray" is created by the set statement that assigns a value to its first element. The for-loop statement prints out the value stored in each element of the array.

Special arguments (command-line parameters)

You can determine the name of the Tcl script file while executing the Tcl script by referring to the $argv0 variable.

```
puts "Executing file $argv0"
```

To access other arguments from the command line, you can use the `lindex` command and the *`argv`* variable:

To read the the Tcl file name:

```
lindex $argv 0
```

To read the first passed argument:

```
lindex $argv 1
```

Example

```
puts "Script name is $argv0" ; # accessing the scriptname
puts "first argument is [lindex $argv 0]"
puts "second argument is [lindex $argv 1]"
puts "third argument is [lindex $argv 2]"
puts "number of argument is [llength $argv]"
set des_name [lindex $argv 0]
puts "Design name is $des_name"
```

Control structures

Tcl control structures are commands that change the flow of execution through a script. These control structures include commands for conditional execution (if-then-elseif-else) and looping (while, for, catch).

An "if" statement only executes the body of the statement (enclosed between curly braces) if the Boolean condition is found to be true.

if/else statements

```
if { “$name” == “paul” } then {
…
# body if name is paul
} elseif { $code == 0 } then {
…
# body if name is not paul and if value of variable code is zero
} else {
…
# body if above conditions is not true
}
```

for loop statement

A "for" statement will repeatedly execute the body of the code as long as the index is within a specified limit.

```
for { set i 0 } { $i < 5 } { incr i } {
…
# body here
}
```

while loop statement

A "while" statement will repeatedly execute the body of the code (enclosed between the curly braces) as long as the Boolean condition is found to be true.

```
while { $p > 0 } {
…
}
```

catch statement

A "catch" statement suspends normal error handling on the enclosed Tcl command. If a variable name is also used, then the return value of the enclosed Tcl command is stored in the variable.

```
catch { open “$inputFile” r } myresult
```

Print statement and Return values

Print Statement

Use the puts command to write a string to an output channel. Predefined output channels are “stdout” and “stderr.” If you do not specify a channel, then puts display text to the stdout channel.

Note: The STDIN Tcl command is not supported by Microsemi SoC tools.

Example:

```
set a [ myprog arg1 arg2 ]
puts "the answer from myprog was $a (this text is on stdout)"
puts stdout “this text also is on stdout”
```

Return Values

The return code of a Tcl command is a string. You can use a return value as an argument to another function by enclosing the command with square brackets [ ].

Example:

```
set a [ prog arg1 arg2 ]
exec $a
```

The Tcl command “exec” will run an external program. The return value of “exec” is the output (on stdout) from the program.

```
Example:
set tmp [ exec myprog ]
puts stdout $tmp
```

Running Tcl Scripts from the Command Line

You can run Tcl scripts from your Windows or Linux command line as well as pass arguments to scripts from the command line.

To execute a Tcl script file in the Libero SoC Project Manager software from a shell command line:

At the prompt, type the path to the Microsemi SoC software followed by the word "SCRIPT" and a colon, and then the name of the script file as follows:

```
<location of Microsemi SoC software>\bin\libero SCRIPT:<filename>
```

where <location of Microsemi SoC software> is the root directory in which you installed the Microsemi SoC software, and <filename> is the name, including a relative or full path, of the Tcl script file to execute. For example, to run the Tcl script file "myscript.tcl", type:

```
C:\libero\designer\bin\libero SCRIPT:myscript.tcl
```

If `myscript.tcl` is in a particular folder named "mydesign", you can use SCRIPT_DIR to change the current working directory before calling the script, as in the following example:

```
C:\libero\designer\bin\libero SCRIPT:myscript.tcl "SCRIPT_DIR:C:\actelprj\mydesign"
```

To pass arguments from the command line to your Tcl script file:

At the prompt, type the path to the Microsemi SoC software followed by the SCRIPT argument:

```
<location of Microsemi SoC software>\bin\designer SCRIPT:<filename "arg1 arg2 ...>" <--
For Libero
```

where `<location of Microsemi SoC software>` is the root directory in which you installed the Microsemi SoC software, and `<filename arg1 arg2 ...>`is the name, including a relative or full path, of the Tcl script file and arguments you are passing to the script file.

For example,

```
C:\libero\designer\bin\designer SCRIPT:myscript.tcl SCRIPT_ARGS:”one two three"
```

To obtain the output from the log file:

At the prompt, type the path to the Microsemi SoC software followed by the SCRIPT and LOGFILE arguments.

```
<location of Microsemi SoC software> SCRIPT:<filename> SCRIPT_ARGS:"a b c"
LOGFILE:<output.log>
```

where

- `location of Microsemi SoC software` is the root directory in which you installed the Microsemi SoC software
- `filename` is the name, including a relative or full path, of the Tcl script file
- `SCRIPT_ARGS` are the arguments you are passing to the script file

- `output.log` is the name of the log file

For example,

```
C:\libero\designer\bin\designer SCRIPT:testTCLparam.tcl SCRIPT_ARGS:"a b c"
LOGFILE:testTCLparam.log
```

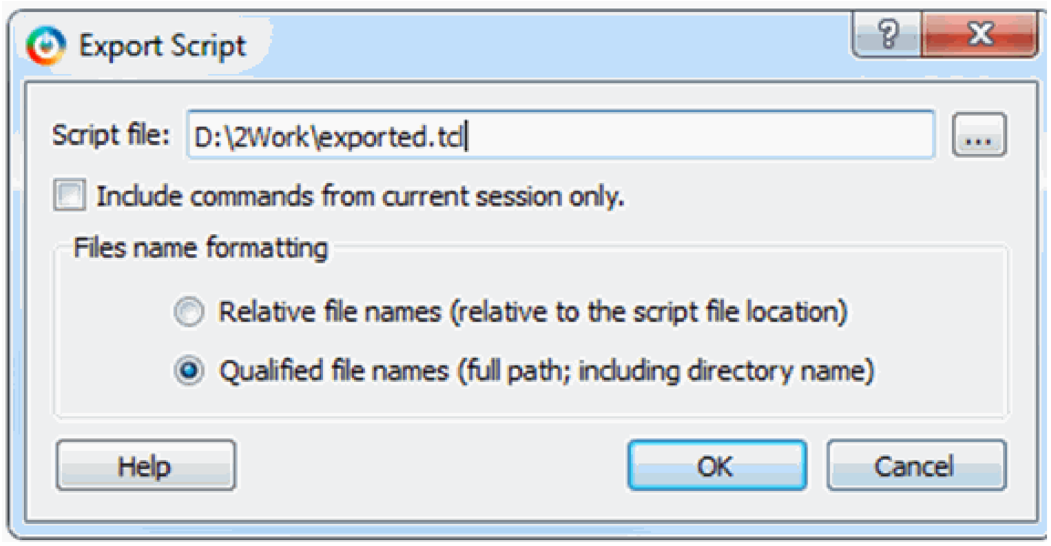
Exporting Tcl Scripts

You can write out a Tcl script file that contains the commands executed in the current session. You can then use this exported Tcl script to re-execute the same commands interactively or in batch. You can also use this exported script to become more familiar with Tcl syntax.

You can export Tcl scripts from the Project Manager.

To export a Tcl session script from the Project Manager:

1. From the **File** menu, choose **Export Script File**. The **Export Script** dialog box appears.
2. Click **OK**. The **Script Export Options** dialog box appears:


Export Script

Script file: D:\2Work\exported.tcl

[ ] Include commands from current session only.

Files name formatting

( ) Relative file names (relative to the script file location)

(●) Qualified file names (full path; including directory name)

Figure 1 · Script Export Options

3. Check the **Include Commands from Current Design [Project] Only** checkbox. This option applies only if you opened more than one design or project in your current session. If so, and you do not check this box, Project Manager exports all commands from your current session.
4. Select the radio button for the appropriate filename formatting. To export filenames relative to the current working directory, select **Relative filenames (default)** formatting. To export filenames that include a fully specified path, select **Qualified filenames (full path; including directory name)** formatting.

   Choose **Relative filenames** if you do not intend to move the Tcl script from the saved location, or **Qualified filenames** if you plan to move the Tcl script to another directory or machine.
5. Click **OK**.

Project Manager saves the Tcl script with the specified filename.

Note:

- When exporting Tcl scripts, Project Manager always encloses filenames in curly braces to ensure portability.
- Libero SoC software does not write out any Tcl variables or flow-control statements to the exported Tcl file, even if you had executed the design commands using your own Tcl script. The exported Tcl file only contains the tool commands and their accompanying arguments.

extended_run_lib

Note: This is not a Tcl command; it is a shell script that can be run from the command line.

The extended_run_lib Tcl script enables you to run the multiple pass layout in batch mode from a command line.

```
$ACTEL_SW_DIR/bin/libero script:$ACTEL_SW_DIR/scripts/extended_run_lib.tcl
logfile:extended_run.log “script_args:-root path/designer/module_name [-n numPasses] [-
starting_seed_index numIndex] [-compare_criteria value] [-c clockName] [-analysis value] [-
slack_criteria value] [-stop_on_success] [-timing_driven|-standard] [-power_driven value]
[-placer_high_effort value]”
```

Note:

There is no option to save the design files from all the passes. Only the (Timing or Power) result reports from all the passes are saved.

Arguments

`-root path/designer/module_name`

The path to the root module located under the designer directory of the Libero project.

`[-n numPasses]`

Sets the number of passes to run. The default number of passes is 5.

`[-starting_seed_index numIndex]`

Indicates the specific index into the array of random seeds which is to be the starting point for the passes. Value may range from 1 to 100. If not specified, the default behavior is to continue from the last seed index that was used.

`[-compare_criteria value]`

Sets the criteria for comparing results between passes. The default value is set to frequency when the –c option is given or timing constraints are absent. Otherwise, the default value is set to violations.

Value	Description
frequency	Use clock frequency as criteria for comparing the results between passes. This option can be used in conjunction with the -c option (described below).
violations	Use timing violations as criteria for comparing the results between passes. This option can be used in conjunction with the -analysis, -slack_criteria and -stop_on_success options (described below).
power	Use total power as criteria for comparing the results between passes, where lowest total power is the goal.

`[-c clockName]`

Applies only when the clock frequency comparison criteria is used. Specifies the particular clock that is to be examined. If no clock is specified, then the slowest clock frequency in the design in a given pass is used. The clock name should match with one of the Clock Domains in the Summary section of the Timing report.

`[-analysis value]`

Applies only when the timing violations comparison criteria is used. Specifies the type of timing violations (the slack) to examine. The following table shows the acceptable values for this argument:

Value	Description
max	Examines timing violations (slack) obtained from maximum delay analysis. This is the default.

Value	Description
min	Examines timing violations (slack) obtained from minimum delay analysis.

`[-slack_criteria value]`

Applies only when the timing violations comparison criteria is used. Specifies how to evaluate the timing violations (slack). The type of timing violations (slack) is determined by the -analysis option. The following table shows the acceptable values for this argument:

Value	Description
worst	Sets the timing violations criteria to Worst slack. For each pass obtains the most amount of negative slack (or least amount of positive slack if all constraints are met) from the timing violations report. The largest value out of all passes will determine the best pass. This is the default.
tns	Sets the timing violations criteria to Total Negative Slack (tns). For each pass it obtains the sum of negative slack values from the first 100 paths from the timing violations report. The largest value out of all passes determines the best pass. If no negative slacks exist for a pass, then the worst slack is used to evaluate that pass.

`[-stop_on_success]`

Applies only when the timing violations comparison criteria is used. The type of timing violations (slack) is determined by the -analysis option. Stops running the remaining passes if all timing constraints have been met (when there are no negative slacks reported in the timing violations report).

`[-timing_driven|-standard]`

Sets layout mode to timing driven or standard (non-timing driven). The default is -timing_driven or the mode used in the previous layout command.

`[-power_driven value]`

Enables or disables power-driven layout. The default is off or the mode used in the previous layout command. The following table shows the acceptable values for this argument:

Value	Description
off	Does not run power-driven layout.
on	Enables power-driven layout.

`[-placer_high_effort value]`

Sets placer effort level. The default is off or the mode used in the previous layout command. The following table shows the acceptable values for this argument:

Value	Description
off	Runs layout in regular effort.
on	Activates high effort layout mode.

Return

A non-zero value will be returned on error.

Supported Families

PolarFire

Exceptions

None

Place and Route - PolarFire

Multiple Pass Layout - PolarFire

Sample Tcl Script - Project Manager

The following Tcl commands create a new project and set your project options.

```
new_project -location {D:/2Work/my_pf_proj} -name {my_pf_proj} -project_description {}\
-block_mode 0 -standalone_peripheral_initialization 0 -use_enhanced_constraint_flow 1\
-hdl {VERILOG} -family {PolarFire} -die {MPF300TS_ES} -package {FCG1152} -speed {-1} \
-die_voltage {1.0} -part_range {EXT} -adv_options {IO_DEFT_STD:LVCMOS 1.8V}\
-adv_options {RESTRICTPROBEPINS:1} -adv_options {RESTRICTSPIPINS:0}\
-adv_options {SYSTEM_CONTROLLER_SUSPEND_MODE:1} -adv_options {TEMPR:EXT} \
-adv_options {VCCI_1.2_VOLTR:EXT} -adv_options {VCCI_1.5_VOLTR:EXT} \
-adv_options {VCCI_1.8_VOLTR:EXT} -adv_options {VCCI_2.5_VOLTR:EXT} \
-adv_options {VCCI_3.3_VOLTR:EXT} -adv_options {VOLTR:EXT}
#Import HDL source file
import_files -convert_EDN_to_HDL 0 -hdl_source {C:/test/prep1.v}
#Import HDL stimulus file
import_files -convert_EDN_to_HDL 0 -stimulus {C:/test/prep1tb.v}
#set the top level design name
set_root -module {prep1::work}
#Associate SDC constraint file to Place and Route tool
organize_tool_files -tool {PLACEROUTE} -file {D:/2Work/my_pf_proj/constraint/user.sdc} \
-module {prep1::work} -input_type {constraint}
#Associate SDC constraint file to Verify Timing tool
organize_tool_files -tool {VERIFYTIMING} -file
{D:/2Work/my_pf_proj/constraint/user.sdc}\
 -module {prep1::work} -input_type {constraint}
#Run synthesize
run_tool -name {SYNTHESIZE}
#Configure Place and Route tool
configure_tool -name {PLACEROUTE} -params {DELAY_ANALYSIS:MAX} -params
{EFFORT_LEVEL:false}\
 -params {INCRPLACEANDROUTE:false} -params {MULTI_PASS_CRITERIA:VIOLATIONS}\
 -params {MULTI_PASS_LAYOUT:false} -params {NUM_MULTI_PASSES:5} -params {PDPR:false}\
 -params {RANDOM_SEED:0} -params {REPAIR_MIN_DELAY:false} -params
{SLACK_CRITERIA:WORST_SLACK} \
 -params {SPECIFIC_CLOCK:} -params {START_SEED_INDEX:1} -params
{STOP_ON_FIRST_PASS:false}\
 -params {TDPR:true}
#Run Place and Route
run_tool -name {PLACEROUTE}
#Configure Timing Report Generation
configure_tool -name {VERIFYTIMING} -run_tool -name {PLACEROUTE}params
{CONSTRAINTS_COVERAGE:1}\
-params {FORMAT:XML} -params {MAX_TIMING_FAST_HV_LT:0} -params {MAX_TIMING_SLOW_LV_HT:1}
\
-params {MAX_TIMING_SLOW_LV_LT:0} -params {MAX_TIMING_VIOLATIONS_FAST_HV_LT:0} \
-params {MAX_TIMING_VIOLATIONS_SLOW_LV_HT:1} -params
{MAX_TIMING_VIOLATIONS_SLOW_LV_LT:0}\
-params {MIN_TIMING_FAST_HV_LT:1} -params {MIN_TIMING_SLOW_LV_HT:0} -params
```

```
{MIN_TIMING_SLOW_LV_LT:0} -params {MIN_TIMING_VIOLATIONS_FAST_HV_LT:1} -params
{MIN_TIMING_VIOLATIONS_SLOW_LV_HT:0} \
-params {MIN_TIMING_VIOLATIONS_SLOW_LV_LT:0}

 #Run Verify Timing tool
run_tool -name {VERIFYTIMING}
#Run Power Verification tool
run_tool -name {VERIFYPOWER}
#Export bitstream
export_bitstream_file -file_name {prep1} \
-export_dir {D:\2Work\my_pf_proj\designer\prep1\export} -format {STP} -master_file 0 \
-master_file_components {} -encrypted_uek1_file 0 -encrypted_uek1_file_components {} \
-encrypted_uek2_file 0 -encrypted_uek2_file_components {} \
-trusted_facility_file 1 -trusted_facility_file_components {FABRIC}
```

Project Manager Tcl Commands

add_file_to_library

Tcl command; adds a file to a library in your project.

```
add_file_to_library
-library name
-file name
```

Arguments

`-library name`

Name of the library where you wish to add your file.

`-file name`

Specifies the new name of the file you wish to add (must be a full pathname).

Example

Add a file named foo.vhd from the ./project/hdl directory to the library 'my_lib'

```
add_file_to_library -library my_lib -file ./project/hdl/foo.vhd
```

See Also

add_library
remove_library
rename_library

add_library

Tcl command; adds a VHDL library to your project.

```
add_library
-library name
```

Arguments

`-library name`

Specifies the name of your new library.

Example

Create a new library called 'my_lib'.

```
add_library –library my_lib
```

See Also

remove_library
rename_library

add_modelsim_path

Tcl command; adds a ModelSim simulation library to your project.

```
add_modelsim_path -lib library_name [-path library_path] [-remove " "]
```

Arguments

`-lib library_name`

Name of the library you want to add.

`-path library_path`

Path to library that you want to add.

`-remove " "`

Name of library you want to remove (if any).

Example

Add the ModelSim library 'msim_update2' located in the c:\modelsim\libraries directory and remove the library 'msim_update1':

```
add_modelsim_path -lib msim_update2 [-path c:\modelsim\libraries] [-remove msim_update1]
```

add_profile

Tcl command; sets the same values as the Add or Edit Profile dialog box. The newly added profile becomes the active tool profile for the specified *type* of tool.

```
add_profile -name profilename -type value -tool profiletool -location tool_location [-args
tool_parameters] [-batch value]
```

Arguments

`-name profilename`

Specifies the name of your new profile.

`-type value`

Specifies your profile type, where value is one of the following:

Value	Description
synthesis	New profile for a synthesis tool
simulation	New profile for a simulation tool
stimulus	New profile for a stimulus tool
flashpro	New FlashPro tool profile

`-tool profiletool`

Name of the tool you are adding to the profile.

`-location tool_location`

Full pathname to the location of the tool you are adding to the profile.

`-args tool_parameters`

Profile parameters (if any).

`-batch value`

Runs the tool in batch mode (if TRUE). Possible values are:

Value	Description
TRUE	Runs the profile in batch mode

Value	Description
FALSE	Does not run the profile in batch mode

Example

Create a new Synthesis tool profile called 'synpol' linked to a Synplify Pro ME installation in my /sqatest/bin directory

```
add_profile -type synthesis -name synpol -tool "Synplify Pro ME" -location
"/sqatest9/bin/synplify_pro" -batch FALSE
```

associate_stimulus

Tcl command; associates a stimulus file in your project.

```
associate_stimulus
[-file name]*
[-mode value]
-module value
```

Arguments

`-file name`

Specifies the name of the file to which you want to associate your stimulus files.

`-mode value`

Specifies whether you are creating a new stimulus association, adding, or removing; possible values are:

Value	Description
new	Creates a new stimulus file association
add	Adds a stimulus file to an existing association
remove	Removes an stimulus file association

`-module value`

Sets the module, where value is the name of the module.

Example

The example associates a new stimulus file 'stim.vhd' for stimulus.

```
associate_stimulus -file stim.vhd -mode new -module stimulus
```

change_link_source

Tcl command; changes the source of a linked file in your project.

```
change_link_source -file filename -path new_source_path
```

Arguments

`-file filename`

Name of the linked file you want to change.

`-path new_source_path`

Location of the file you want to link to.

Example

Change the link to a file 'sim1.vhd' in your project and link it to the file in c:\microsemi\link_source\simulation_test.vhd

```
change_link_source -file sim1.vhd -path c:\microsemi\link_source\simulation_test.vhd
```

check_fdc_constraints

This Tcl command checks FDC constraints files associated with the Synthesis tool.

```
check_fdc_constraints –tool {synthesis}
```

Arguments

`-tool {synthesis}`

Example

```
check_fdc_constraints –tool {synthesis}
```

Return Value

This command returns “0” on success and “1” on failure.

check_hdl

Tcl command; checks the HDL in the specified file.

```
check_hdl -file filename
```

Arguments

`-file filename`

Name of the HDL file you want to check.

Example

Check HDL on the file hdl1.vhd.

```
check_hdl -file hdl1.vhd
```

check_ndc_constraints

This Tcl command checks NDC constraints files associated with the Synthesis tool. NDC constraints are used to optimize the post-synthesis netlist with the Libero SoC Compile engine.

```
check_ndc_constraints –tool {synthesis}
```

Arguments

-tool {synthesis}

Example

```
check_ndc_constraints –tool {synthesis}
```

check_pdc_constraints

This Tcl command checks PDC constraints files associated with the Libero Place and Route tool.

```
check_pdc_constraints –tool {designer}
```

Arguments

```
-tool {designer}
```

Example

```
check_pdc_constraints –tool {designer}
```

Return Value

This command returns “0” on success and “1” on failure.

check_sdc_constraints

This Tcl command checks SDC constraints files associated with the Libero tools: designer, synthesis, or timing.

```
check_sdc_constraints –tool {tool_name}
```

Arguments

```
-tool {synthesis|designer|timing}
```

Example

This command checks the SDC constraint files associated with Timing Verifcation.

```
check_sdc_constraints –tool {timing}
```

This command checks the SDC constraint files associated with Place and Route.

```
check_sdc_constraints –tool {designer}
```

This command checks the SDC constraint files associated with Synthesis.

```
check_sdc_constraints –tool {synthesis}
```

Return Value

The command returns “0” on success and “1” on failure.

close_design

Tcl command; closes the current design and brings Designer to a fresh state to work on a new design. This is equivalent to selecting the Close command from the File menu.

```
close_design
```

Arguments

None

Supported Families

See the Tcl Commands and Supported Families table for the list of families that support this command.

Example

```
if  { [catch { close_design }] {
              Puts “Failed to close design”
              # Handle Failure
} else {
              puts “Design closed successfully”
              # Proceed with processing a new design
}
```

See Also

open_design
close_design
new_design
Designer Tcl Command Reference

close_project

Tcl command; closes the current project in Libero SoC. Equivalent to clicking the File menu, and choosing Close Project.

```
close_project
```

Arguments

None

Example

```
close_project
```

See Also

open_project

configure_tool

configure_tool is a general-purpose Tcl command that is used to set the parameters for any tool called by Libero. The command requires the name of the tool and one or more parameters in the format *tool_parameter*:*value*. These parameters are separated and passed to the tool to set up its run.

```
configure_tool
-name {<tool_name>} # Each tool_name has its own set of parameters
-params {<parameter>:<value>} # List of parameters and values
tool_name ::=  CONFIGURE_PROG_OPTIONS | SYNTHESIZE | PLACEROUTE |
GENERATEPROGRAMMINGFILE | PROGRAMDEVICE |  PROGRAMMER_INFO |IO_PROGRAM_STATE | SPM |
VERIFYTIMING  | PROGRAM_SPI_FLASH_IMAGE
```

Supported tool_names

The following table lists the supported tool_names.

tool_name	Parameter (-params)	Description
"CONFIGURE_PROG_OPTIONS " on page **Error! Bookmark not defined.**	See the topic for parameter names and values.	See the topic for description.

tool_name	Parameter (-params)	Description
SYNTHESIZE	See the topic for parameter names and values.	See the topic for description.
PLACEROUTE	See the topic for parameter names and values.	See the topic for description.
"GENERATEPROGRAMMINGFILE" on page **Error! Bookmark not defined.**	See the topic for parameter names and values.	See the topic for description.
PROGRAMDEVICE	See the topic for parameter names and values.	See the topic for description.
PROGRAMMER_INFO	See the topic for parameter names and values.	See the topic for description.
IO_PROGRAMMING_STATE	See the topic for parameter names and values.	See the topic for description.
SPM	See the topic for parameter names and values.	See the topic for description.
VERIFYTIMING	See the topic for parameter names and values.	See the topic for description.
"PROGRAM_SPI_FLASH_IMAGE" on page **Error! Bookmark not defined.**	See the topic for parameter names and values.	See the topic for description.

See Also

Tcl documentation conventions

create_links

Tcl command; creates a link (or links) to a file/files in your project.

```
create_links [-hdl_source file]* [-stimulus file]* [-sdc file]* [-pin file]* [-dcf file]* [-
gcf file]* [-pdc file]* [-crt file]* [-vcd file]*
```

Arguments

`-hdl_source file`

Name of the HDL file you want to link.

`-stimulus file`

Name of the stimulus file you want to link.

`-sdc file`

Name of the SDC file you want to link.

`-pin file`

Name of the PIN file you want to link.

`-dcf file`

Name of the DCF file you want to link.

`-gcf file`

Name of the GCF file you want to link.

`-pdc` *`file`*

Name of the PDC file you want to link.

`-crt` *`file`*

Name of the crt file you want to link.

`-vcd` *`file`*

Name of the VCD file you want to link.

Example

Create a link to the file hdl1.vhd.

```
create links [-hdl_source hdl1.vhd]
```

delete_files

Tcl command; deletes files in your Libero SoC project.

```
delete_files
-file value
-from_disk
```

Arguments

`-file` *`value`*

Specifies the file you wish to delete from the project. This parameter is required for this Tcl command. It does not delete the file from the disk. Use the -from_disk flag to delete a file from the disk. Value is the name of the file you wish to delete (including the full pathname).

`-from_disk`

Deletes a file from the disk.

Example

Delete the files file1.vhd and file2.vhd from the project, and delete the file top_palace.sdc from the disk.

```
delete_files –file ./project/hdl/file1.vhd –file ./project/hdl/file2.vhd
delete_files –from_disk –file ./project/phy_synthesis/top_palace.sdc
```

The following command deletes the core 'add1' from your disk and project (it is the same as the command to delete an IP core from your disk and project).

```
delete_files -from_disk -file ./project/component/work/add1/add1.cxf
```

See Also

close_project

new_project

download_core

Tcl command; downloads a core and adds it to your repository.

```
download_core [-vlnv "vlnv"]+ [-location "location"]
```

Arguments

`-vlnv` *`vlnv`*

Vendor, library, name and version of the core you want to download.

`-location` *`core_name`*

Location of the repository where you wish to add the core.

Example

Download the core CoreAXI to the repository www.actel-ip.com/repositories/SgCore:

```
download_core -vlnv {Actel:SystemBuilder:PF_DDR4:1.0.102} -location {www.actel-
ip.com/repositories/SgCore}
```

edit_profile

Tcl command; sets the same values as the Add or Edit Profile dialog box.

```
edit_profile -name profilename -type value -tool profiletool -location profilelocation [-args
parameters] [-batch value] [-new_name name]
```

Arguments

`-name profilename`

Specifies the name of your new profile.

`-type value`

Specifies your profile type, where value is one of the following:

Value	Description
synthesis	New profile for a synthesis tool
simulation	New profile for a simulation tool
stimulus	New profile for a stimulus tool
flashpro	New FlashPro tool profile

`-tool profiletool`

Name of the tool you are adding to the profile.

`-location profilelocation`

Full pathname to the location of the tool you are adding to the profile.

`-args parameters`

Profile tool parameters (if any).

`-batch value`

Runs the tool in batch mode (if TRUE). Possible values are:

Value	Description
TRUE	Runs the profile in batch mode
FALSE	Does not run the profile in batch mode

`-new_name name`

Name of new profile.

Example

Edit a FlashPro tool profile called 'myflashpro' linked to a new FlashPro installation in my c:\programs\actel\flashpro\bin directory, change the name to updated_flashpro.

```
edit_profile -name myflashpro -type flashpro -tool flashpro.exe -location
c:\programs\actel\flashpro\bin\flashpro.exe -batch FALSE -new_name updated_flashpro
```

export_as_link

Tcl command; exports a file to another directory and links to the file.

```
export_as_link -file filename -path link_path
```

Arguments

`-file filename`

Name of the file you want to export as a link.

`-path link_path`

Path of the link.

Example

Export the file hdl1.vhd as a link to c:\microsemi\link_source.

```
export_as_link -file hdl1.vhd -path c:\microsemi\link_source
```

export_bitstream_file

Configures the parameters for the bitstream to be exported from Libero.

```
 export_bitstream_file
[-file_name file]
[-export_dir dir]
[-format STP |  DAT | SPI]
[-force_rtg4_otp 0 | 1]
[-master_file 0 | 1]
[-master_file_components SECURITY | FABRIC  | SNVM]
[-encrypted_uek1_file 1 | 0]
[-encrypted_uek1_file_components FABRIC | SNVM]
[-encrypted_uek2_file 1 | 0]
[-encrypted_uek2_file_components FABRIC  | SNVM]
[-trusted_facility_file 1 | 0]
[-trusted_facility_file_components FABRIC  | SNVM]
```

Arguments

`-file_name file`

The name of the file. File name must start with design name. If omitted, design name will be used.

`-export_dir dir`

Location where the bitstream file will be exported. If omitted, design export folder will be used.

`-format STP  | DAT | SPI`

Specifies the bitstream file formats to be exported. Space is used as a delimiter. If omitted, STP file will be exported.

`-force_rtg4_otp 0 | 1`

Enforces the use of One-time programming (OTP).

Security-related options:

Note: One of the bitstream files must be set to “1”. 1 indicates that this particular file type will be imported; 0 indicates that it will not be exported. For example, if `trusted_facility_file` is set to 1, all other file types must be set to 0.

Or, if `trusted_facility_file` is set to 0, a combination of `master_file` and `euk1_file` and `euk2_file` can be set to 1. In this case, `master_file` must be set to 1.

Bitstream encryption with default key (default security):

`-trusted_facility_file 1 | 0`

Specifies the bitstream file to be exported.

`-trusted_facility_file_components` *`FABRIC  | SNVM`*

Specifies the components of the design that will be saved to the bitstream file. The value can be any one or both of FABRIC and SNVM.

Custom security options:

`-master_file` *`0 | 1`*

Specifies the bitstream files to be exported. Depends on the selected security.

Note: If `-master_file` is 1, SECURITY must be selected.

`-master_file_components` *`SECURITY | FABRIC  | SNVM`*

Specifies the components in the design that will be saved to the bitstream file. The value can be any one or any combination of SECURITY, FABRIC, SNVM.

`-encrypted_uek1_file` *`0 | 1`*

`-encrypted_uek1_file_components` *`FABRIC  | SNVM`*

Specifies the components of the design that will be saved to uek1 bitstream. The value can be any one or both of FABRIC and SNVM.

`-encrypted_uek2_file` *`0 | 1`*

`-encrypted_uek2_file_components` *`FABRIC  | SNVM`*

Specifies the components of the design that will be saved to uek2 bitstream. The value can be any one or both of FABRIC and SNVM.

Bitstream file to be exported and the components of the design that will be saved to the bitstream file are required.

Note: A TCL script file exported from Libero will include all command options. You can modify options you need and remove options you do not need.

Supported Families

See the Tcl Commands and Supported Families table for the list of families that support this command.

Example

Export a bitstream file:

Export bitstream file for design with default security

```
export_bitstream_file \
  -trusted_facility_file 1
  -trusted_facility_file_components {FABRIC}
```

Export bitstream file for design with custom security options

Export bitstream to master file with FABRIC component.

```
export_bitstream_file \
  -master_file 1 \
  -master_file_components {FABRIC }
```

Export bitstreams to master and uek1 encrypted files. Master file to include security and fabric components, uek1 encrypted file to include FABRIC.

```
export_bitstream_file\
  -file_name {iaptcl1}\
  -format {STP}\
  -master_file 1\
  -master_file_components {SECURITY FABRIC}\
  -encrypted_uek1_file 1\
  -encrypted_uek1_file_components {FABRIC}\
  -encrypted_uek2_file 0\
  -encrypted_uek2_file_components { }\
```

export_bsdl_file

Tcl command to export the BSDL to a specified file. The exported file has a *.bsd file name extension.

```
export_bsdl_file
-file {absolute path and name of BSDL file}
```

Arguments

`-file {absolute path and name of BSDL file}`

Specifies the *.bsd file.

Returns

Returns 0 on success, 1 on failure.

Example

```
export_bsdl_file\
  -file {E:/designs/export/sd1.bsd}
```

export_design_summary

This Tcl command exports an HTML file containing information about your root SmartDesign in your project. The HTML report provides information on:

- Generated Files
- I/Os
- Hardware Instances
- Firmware
- Memory Map

```
export_design_summary –file {D: /Designs/test/sd1.html}
```

Returns

Returns 0 on success, 1 on failure.

export_netlist_file

Tcl command to export the netlist after the compile state has completed. The netlist can be either Verilog or VHDL. Microsemi recommends exporting the netlist after the compile state has successfully completed.

```
export_netlist_file
-file {absolute path and filename for netlist}
-vhdl {value}
```

Arguments

`-file {absolute path and filename}`

Specifies the path and name of netlist file.

`-vhdl {value}`

Generates the netlist in VHDL (when set to 1) or Verilog (when set to 0). Default is 0 (Verilog netlist).

Returns

Returns 0 on success, 1 on failure.

Example

```
export_netlist_files\
  -file {E:/designs/export/sd1/sd1.v}\
  -vhdl 0
```

export_pin_reports

Tcl command to configure and export a pin report file to a specified folder/directory location.

```
export_pin_reports
-export_dir {absolute path to folder location}
-pin_report_by_name {value}
-pin_report_by_pkg_pin {value}
-bank_report {value}}
-io_report {value}
```

Arguments

`-export_dir {absolute or relative path to the folder for pin report file}`

Specifies the folder.

`-pin_report_by_name {value}`

Set to 1 to have the pin report sorted by pin name. Default is 1.

`- pin_report_by_pkg_pin {value}`

Set to 1 to have pin report sorted by package pin number, 0 to not sort by package pin number. Default is 1.

`- bank_report {value}`

Set to 1 to generate the I/O bank report, 0 to not generate the report. Default is 1.

`- io_report {value}`

Set to 1 to generate the I/O report, 0 to not generate the report. Default is 1.

At least one argument must be specified for this command.

Returns

Returns 0 on success, 1 on failure.

Example

```
export_pin_reports\
  -export_dir {E:/designs/export}\
  -pin_report_by_name {1}\
  -pin_report_by_pkg_pin {0}\
  -bank_report {1}\
  -io_report {1}
```

export_profiles

Tcl command; exports your tool profiles. Performs the same action as the Export Profiles dialog box.

```
export_profile -file name [-export value]
```

Arguments

`-file name`

Specifies the name of your exported profile.

`-export value`

Specifies your profile export options. The following table shows the acceptable values for this argument:

Value	Description
predefined	Exports only predefined profiles
user	Exports only user profiles
all	Exports all profiles

Example

The following command exports all profiles to the file 'all_profiles':

```
export_profiles -file all_profiles [-export all]
```

export_script

Tcl command; export_script is a command that explicitly exports the Tcl command equivalents of the current Libero session. You must supply a file name with the -file parameter. You may supply the optional -relative_path parameter to specify whether an absolute or relative path is used in the exported script file.

```
export_script\
-file {<absolute or relative path to constraint file>} \
-relative_path <value> \
```

Arguments

`-file {<absolute or relative path to constraint file>}`

Specifies the absolute or relative path to the constraint file; there may be multiple -file arguments (see example below).

`-relative_path {<value>}`

Sets your option to use a relative or absolute path in the exported script; use 1 for relative path, 0 for absolute.

Example

```
export_script -file {./exported.tcl} -relative_path 1
```

generate_sdc_constraint_coverage

Tcl command to generate the constraint coverage report. The constraint coverage report contains information about the coverage of the paths from associated SDC constraints in the design. Two constraints coverage reports can be generated, one for Place and Route and one for Timing Verification.

This command is available for the Enhanced Constraint Flow only. To run this command, there is no need to run Place-and-Route first, but the design must be in the post-synthesis state. The generated constraint coverage reports (*.xml) are listed in the Reports tab and are physically located in <prj_folder>/designer/<module>/*constraints_coverage.xml.

```
generate_sdc_constraint_coverage –tool {PLACEROUTE | VERIFYTIMING}
```

Arguments

```
-tool {PLACEROUTE|VERIFYTIMING}
```

Specifies whether the constraint coverage report is based on the SDC constraint file associated with Place and Route or associated with Timing Verification.

Returns

Returns 0 on success, 1 on failure.

Example

This command generates the SDC Constraint Coverage report for the SDC file associated with Place and Route:

```
generate_sdc_constraint_coverage –tool {PLACEROUTE}
```

This command generates the SDC Constraint Coverage report for the SDC file associated with Timing Verification:

```
generate_sdc_constraint_coverage –tool {VERIFYTIMING}
```

See Also

Understanding Constraints Coverage Reports

import_files (Libero SoC)

Tcl command; enables you to import design source files and constraint files.

SmartFusion2, IGLOO2, RTG4, and PolarFire only: For importing constraint files, import_files has retired the -pdc parameter for SmartFusion2 and IGLOO2. It has been replaced with two new parameters to match the new design flow. Physical Design Constraints (PDC) Tcl must now be divided between I/O attribute and pin information from all floorplanning and timing constraints. These commands must now reside in and be imported as separate files. The new parameters specify the type of *.pdc file being imported.

Use of the -pdc parameter with Smartfusion2 or IGLOO2 families will cause an error. The path to the file can be absolute or relative but must be enclosed in curly braces { }.

Use the -can_convert_EDN_to_HDL parameter to convert the EDIF file to HDL and then import the converted HDL file.

Note: The EDIF File is not imported.

```
import_files

-smartgen_core {file}
-ccp {file}
-stimulus {file}
-hdl_source {file}
-io_pdc {<absolute or relative path to file>} # For PDC containing I/O attribute and pin info
-fp_pdc {<absolute or relative path to file>} # For PDC containing timing and placement info
-edif {file}
-sdc {file}
-pin {file}
-dcf {file}
-pdc {file}

-vcd {file}
-saif {file}
-crt {file}
-simulation {file}
-profiles {file}
-cxf {file}
```

```
-templates {file}
-ccz {file}
-wf_stimulus {file}
-modelsim_ini {file}
-library {file}
-can_convert_EDN_to_HDL {true | false
```

Arguments

`-smartgen_core {file}`

Specifies the cores you wish to import into your project. Type parameter must be repeated for each file.

`-ccp {file}`

Specifies the ARM or Cortex-M1 cores you wish to import into your project. Type parameter must be repeated for each file.

`-stimulus {file}`

Specifies HDL stimulus files you wish to import into your project. Type parameter must be repeated for each file.

`-hdl_source {file}`

Specifies the HDL source files you wish to import into your project. Type parameter must be repeated for each file.

`-io_pdc {<absolute or relative path to file>}`

SmartFusion2 and IGLOO2 only - Specifies the PDC file that contains the I/O attribute and pin information.

`-fp_pdc {<absolute or relative path to file>}`

SmartFusion2 and IGLOO2 only - Specifies the PDC file that contains the timing and placement information.

`-edif {file}`

Specifies the EDIF files you wish to import into your project. Type parameter must be repeated for each file. This is a mandatory option if you want to convert EDIF to HDL with the –can_convert_EDN_to_HDL option.

`-can_convert_EDN_to_HDL {true |false |1 | 0} #Boolean {true | false | 1 | 0}`

The –edif option is mandatory. If the –edif option is not specified or the –can_convert_EDN_to_HDL is used with another option, EDIF to HDL conversion will fail.

`-constraint_sdc {file}`

Specifies the SDC constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_pin {file}`

Specifies the PIN constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_dcf {file}`

Specifies the DCF constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_pdc {file}`

Specifies the PDC constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_gcf {file}`

Specifies the GCF constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_vcd {file}`

Specifies the VCD constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_saif {file}`

Specifies the SAIF constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_crt {file}`

Specifies the CRT constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-simulation {file}`

Specifies the simulation files you wish to import into your Libero SoC project. Type parameter must be repeated for each file.

`-profiles {file}`

Specifies the profile files you wish to import into your Libero SoC project. Type parameter must be repeated for each file.

`-cxf {file}`

Specifies the CXF file (such as SmartDesign components) you wish to import into your Libero SoC project. Type parameter must be repeated for each file.

`-templates {file}`

Specifies the template file you wish to import into your IDE project.

`-ccz {file}`

Specifies the IP core file you wish to import into your project.

`-wf_stimulus {file}`

Specifies the WaveFormer Pro stimulus file you wish to import into your project.

`-modelsim_ini {file}`

Specifies the ModelSIM INI file that you wish to import into your project.

`-library {file}`

Specifies the library file that you wish to import into your project. If a library file is not available it will be created and added to the library.

Example

The command below imports the HDL source files file1.vhd and file2.vhd:

```
import_files -hdl_source file1.vhd –hdl_source file2.vhd
```

new_project

Tcl command; creates a new project in Libero SoC. If you do not specify a location, Libero SoC saves the new project in your current working directory.

```
new_project -name project_name\

-location project_location -family family_name\
-project_description brief text description of project\
–die device_die -package package_name -hdl HDL_type\
-speed speed_grade -die_voltage value\
-ondemand_build_dh {1 | 0} \
-adv_options value\
```

Arguments

`-name project_name`

The name of the project. This is used as the base name for most of the files generated from Libero SoC.

`-location project_location`

The location of the project. Must not be an existing directory.

`-project_description project_description`

A brief text description of the design in your project.

`-family` *`family_name`*

The Microsemi SoC device family for your targeted design.

`-die` *`device_die`*

Die for your targeted design.

`-package` *`package_name`*

Package for your targeted design.

`-hdl` *`HDL_type`*

Sets the HDL type for your new project.

Value	Description
VHDL	Sets your new projects HDL type to VHDL
VERILOG	Sets your new projects to Verilog

`-speed` *`speed_grade`*

Sets the speed grade for your project. Possible values depend on your device, die and package. See your device datasheet for details.

`-die_voltage` *`value`*

Sets the die voltage for your project. Possible values depend on your device. See your device datasheet for details.

`-ondemand_build_dh {1 | 0}`

Enter "1" to enable or "0" (default) to disable On Demand Build Design Hierarchy.

`-adv_options` *`value`*

Sets your advanced options, such as operating conditions.

Value	Description
IO_DEFT_STD:LVTTL	Sets your I/O default value to LVTTL. This value defines the default I/O technology to be used for any I/Os that the user does not explicitly set a technology for in the I/O Editor. It could be any of : • LVTTL • LVCMOS 3.3V • LVCMOS 2.5V • LVCMOS 1.8V • LVCMOS 1.5V • LVCMOS 1.2V
RESTRICTPROBEPINS	This value reserves your pins for probing if you intend to debug using SmartDebug. Two values are available: • 1 (Probe pins are reserved) • 0 (No probe pins are reserved)
SYSTEM_CONTROLLER_SUSPEND_MODE	Enables designers to suspend operation of the System Controller. Enabling this bit instructs the System Controller to place itself in a reset state once the device is powered up. This effectively suspends all system services from being performed. For a list of system services, refer to the PolarFire FPGA Fabric User Guide for your device on the Microsemi website. Two values are available:

Value	Description
	• 1 (System Controller Suspend Mode is enabled) • 0 (System Controller Suspend Mode is disabled
The following options are for Analysis Operating Conditions so that Timing and Power analysis can be performed at different operating conditions.	
TEMPR	Sets your default temperature range for operating condition analysis to EXT or IND
VCCI_1.2_VOLTR	Sets the Default I/O Voltage Range for 1.2V to EXT or IND These settings are propagated to Verify Timing, Verify Power and Backannotated Netlist to perform Timing/Power Analysis
VCCI_1.5_VOLTR	Sets the Default I/O Voltage Range for 1.5V to EXT or IND These settings are propagated to Verify Timing, Verify Power and Backannotated Netlist to perform Timing/Power Analysis
VCCI_1.8_VOLTR	Sets the Default I/O Voltage Range for 1.8V to EXT or IND These settings are propagated to Verify Timing, Verify Power and Backannotated Netlist to perform Timing/Power Analysis
VCCI_2.5_VOLTR	Sets the Default I/O Voltage Range for 2.5V to EXT or IND These settings are propagated to Verify Timing, Verify Power and Backannotated Netlist to perform Timing/Power Analysis
VCCI_3.3_VOLTR	Sets the Default I/O Voltage Range for 3.3V to EXT or IND These settings are propagated to Verify Timing, Verify Power and Backannotated Netlist to perform Timing/Power Analysis
VOLTR	Sets the core voltage range for operating condition analysis to EXT or IND. This setting is propagated to Verify Timing, Verify Power and Backannotated Netlist to perform Timing/Power Analysis.
PART_RANGE	Sets your default temperature range for your project to EXT or IND.

```
#Create a new project and set up a new design
new_project -location {D:/2Work/my_pf_proj} -name {my_pf_proj} -project_description {}\-
block_mode 0 -standalone_peripheral_initialization 0 -use_enhanced_constraint_flow 1\-
hdl {VERILOG} -family {PolarFire} -die {MPF300TS_ES} -package {FCG1152} -speed {-1} \-
die_voltage {1.0} -part_range {EXT} -adv_options {IO_DEFT_STD:LVCMOS 1.8V}\-adv_options
{RESTRICTPROBEPINS:1} -adv_options {RESTRICTSPIPINS:0}\-adv_options
{SYSTEM_CONTROLLER_SUSPEND_MODE:1} -adv_options {TEMPR:EXT} \-adv_options
{VCCI_1.2_VOLTR:EXT} -adv_options {VCCI_1.5_VOLTR:EXT} \-adv_options
{VCCI_1.8_VOLTR:EXT} -adv_options {VCCI_2.5_VOLTR:EXT} \-adv_options
{VCCI_3.3_VOLTR:EXT} -adv_options {VOLTR:EXT}
#Import HDL source file
import_files -convert_EDN_to_HDL 0 -hdl_source {C:/test/prep1.v}
#Import HDL stimulus file
import_files -convert_EDN_to_HDL 0 -stimulus {C:/test/prep1tb.v}
#set the top level design name
set_root -module {prep1::work}
```

```
#Associate SDC constraint file to Place and Route tool
organize_tool_files -tool {PLACEROUTE} -file {D:/2Work/my_pf_proj/constraint/user.sdc}
\-module {prep1::work} -input_type {constraint}
#Associate SDC constraint file to Verify Timing tool
organize_tool_files -tool {VERIFYTIMING} -file
{D:/2Work/my_pf_proj/constraint/user.sdc}\ -module {prep1::work} -input_type
{constraint}
#Run synthesize
run_tool -name {SYNTHESIZE}
#Configure Place and Route tool
configure_tool -name {PLACEROUTE} -params {DELAY_ANALYSIS:MAX} -params
{EFFORT_LEVEL:false}\ -params {INCRPLACEANDROUTE:false} -params
{MULTI_PASS_CRITERIA:VIOLATIONS}\ -params {MULTI_PASS_LAYOUT:false} -params
{NUM_MULTI_PASSES:5} -params {PDPR:false}\ -params {RANDOM_SEED:0} -params
{REPAIR_MIN_DELAY:false} -params {SLACK_CRITERIA:WORST_SLACK} \ -params
{SPECIFIC_CLOCK:} -params {START_SEED_INDEX:1} -params {STOP_ON_FIRST_PASS:false}\ -
params {TDPR:true}
```

open_project

Tcl command; opens an existing Libero SoC project.

```
open_project project_name-do_backup_on_convert value-backup_file backup_filename
```

Arguments

project_name

Must include the complete path to the PRJ file. If you do not provide the full path, Libero SoC infers that you want to open the project from your current working directory.

`-do_backup_on_convert` *value*

Sets the option to backup your files if you open a project created in a previous version of Libero SoC.

Value	Description
TRUE	Creates a backup of your original project before opening
FALSE	Opens your project without creating a backup

`-backup_file` *backup_filename*

Sets the name of your backup file (if you choose to do_backup_on_convert).

Example

Open project.prj from the c:/netlists/test directory.

```
open_project c:/netlists/test/project.prj
```

See Also

close_project
new_project
save_project

organize_constraints

Tcl command; organizes the constraint files in your project.

```
-organize_constraints
[-file name]*
[-mode value]
-designer_view name
-module value
-tool value
```

Arguments

`-file name`

Specifies the name of the file to which you want to associate your stimulus files.

`-mode value`

Specifies whether you are creating a new stimulus association, adding, or removing; possible values are:

Value	Description
new	Creates a new stimulus file association
add	Adds a stimulus file to an existing association
remove	Removes an stimulus file association

`-designer_view name`

Sets the name of the Designer View in which you wish to add the constraint file, where name is the name of the view (such as impl1).

`-module value`

Sets the module, where value is the name of the module.

`-tool value`

Identifies the intended use for the file, possible values are:

Value	Description
synthesis	File to be used for synthesis
designer	File to be used in Designer
phsynth	File to be used in physical synthesis

Example

The example adds the constraint file delta.vhd in the Designer View impl2 for the Designer tool.

```
-organize_constraints -file delta.vhd -mode new -designer_view impl2 -module constraint
-tool designer
```

organize_sources

Tcl command; organizes the source files in your project.

Arguments

```
-organize_sources
[-file name]*
```

```
[-mode value]
-module value
-tool value
[-use_default value]
```

Arguments

`-file name`

Specifies the name of the file to which you want to associate your stimulus files.

`-mode value`

Specifies whether you are creating a new stimulus association, adding, or removing; possible values are:

Value	Description
new	Creates a new stimulus file association
add	Adds a stimulus file to an existing association
remove	Removes an stimulus file association

`-module value`

Sets the module, where value is the name of the module.

`-tool value`

Identifies the intended use for the file, possible values are:

Value	Description
synthesis	File to be used for synthesis
simulation	File to be used for simulation

`-use_default value`

Uses the default values for synthesis or simulation; possible values are:

Value	Description
TRUE	Uses default values for synthesis or simulation.
FALSE	Uses user-defined values for synthesis or simulation

Supported Families

See the Tcl Commands and Supported Families table for the list of families that support this command.

Example

The example organizes a new stimulus file 'stim.vhd' using default settings.

```
-organize_sources -file stim.vhd -mode new -module stimulus -tool synthesis -use_default
TRUE
```

See Also

Project Manager Tcl Command Reference

organize_tool_files

This Tcl command is used to specify specific constraint files to be passed to and used by a Libero tool.

```
organize_tool_files \
-tool {tool_name}
-params {tool parameters}
-file {<absolute or relative path to constraint file>} \
-module {$design::work} \
-input_type {value}
```

Arguments

`-tool {<tool_name>}`

Specifies the name of the tool files you want to organize. Valid values are:

`SYNTHESIZE | PLACEROUTE | SIM_PRESYNTH | SIM_POSTSYNTH | SIM_POSTLAYOUT | VERIFYTIMING`

`-file {<absolute or relative path to constraint file>}`

Specifies the absolute or relative path to the constraint file; there may be multiple `-file` arguments (see example below).

`-module {<design::work>}`

Module definition, format is <*$design:work*>.

`-input_type {<constraint>}`

Specifies type of input file. Possible values are: `constraint | source | simulation | stimulus | unknown`

Example

The following command organizes the `test_derived.sdc` and `user.sdc` files of SDC file type for the tool VERIFYTIMING for the sd1: work design.

```
organize_tool_files \
   -tool {VERIFYTIMING} \
   -file {D:/Designs/my_proj/constraints/test_derived.sdc} \
   -file {D:/Designs/my_proj/constraints/user.sdc} \
   -module {sd1::work} \
   -input_type {constraint}
```

project_settings

This Tcl command modifies project flow settings for your Libero SoC project.

```
project_settings [-hdl "VHDL | VERILOG"]\
[-verilog_mode {VERILOG_2K | SYSTEM_VERILOG}] \
[-vhdl_mode {VHDL_2008 | VHDL_93}]\
[-auto_update_modelsim_ini "TRUE | FALSE"]\
[-auto_update_viewdraw_ini "TRUE | FALSE"]\
[-block_mode "TRUE | FALSE"]\

[-auto_generate_synth_hdl "TRUE | FALSE"]\
[-auto_run_drc "TRUE | FALSE"]\
[-auto_generate_viewdraw_hdl "TRUE | FALSE"\
[-auto_file_detection "TRUE | FALSE"]\
[-standalone_peripheral_initialization "1 | 0"]\

[-ondemand_build_dh "1 | 0"]\
[-enable_design_separation "1 | 0"]\
[-enable_set_mitigation "1 | 0"]\
[-display_fanout_limit {integer}]
```

Arguments

`-hdl "VHDL | VERILOG"`

Sets your project HDL type.

`-verilog_mode {VERILOG_2K | SYSTEM_VERILOG}`

Sets the Verilog standard to Verilog-2001 or System Verilog.

`-vhdl_mode {VHDL_2008 | VHDL_93}`

Sets the VHDL standard to VHDL-2008 or VHDL-1993.

`-auto_update_modelsim_ini "TRUE | FALSE"`

Sets your auto-update modelsim.ini file option. TRUE updates the file automatically.

`-auto_update_viewdraw_ini "TRUE | FALSE"`

Sets your auto-update viewdraw.ini file option. TRUE updates the file automatically.

`-block_mode "TRUE | FALSE"`

Puts the Project Manager in Block mode, enables you to create blocks in your project.

`-auto_generate_synth_hdl "TRUE | FALSE"`

Auto-generates your HDL file after synthesis (when set to TRUE).

`-auto_run_drc "TRUE | FALSE"`

Auto-runs the design rule check immediately after synthesis (when set to TRUE).

`-auto_generate_viewdraw_hdl "TRUE | FALSE"`

Auto-generates your HDL netlist after a Save & Check in ViewDraw (when set to TRUE).

`-auto_file_detection "TRUE | FALSE"`

Automatically detects when new files have been added to the Libero SoC project folder (when set to TRUE).

`-standalone_peripheral_initialization "1 | 0"`

When set to 1, this option instructs System Builder not to build the initialization circuitry for your Peripherals. Set this option to 1 if you want to build your own peripheral initialization logic in SmartDesign to initialize each of the peripherals (MDDR/FDDR/SERDES) independently.

`-ondemand_build_dh "1 | 0"`

Enter "1" to enable or "0" (default) to disable On Demand Build Design Hierarchy.

`-enable_design_separation "1 | 0"`

Set it to “1” if your design is for security and safety critical applications and you want to make your design’s individual subsystems (design blocks) separate and independent (in terms of physical layout and programming) to meet your design separation requirements. When set to “1”, Libero generates a parameter file (MSVT.param) that details design blocks present in the design and the number of signals entering and leaving a design block. Microsemi provides a separate tool, known as Microsemi Separation Verification Tool (MSVT), which checks the final design place and route result against the MSVT.param file and determines whether the design separation meets your requirements.

`-display_fanout_limit {integer}`

Use this option to set the limit of high fanout nets to be displayed; the default value is 10. This means the top 10 nets with the highest fanout will appear in the <root>_compile_netlist.log file.

Example

The following example sets your project to VHDL, disables the auto-update of the ModelSim INI or ViewDraw INI files, enables the auto-generation of HDL after synthesis, enables auto-detection for files, sets the display of high fanout nets to the top 12 high fanout nets, enables SET filters to mitigate radiation-induced transients, and enables design separation methodology for the design.

```
project_settings -hdl "VHDL" \
  -auto_update_modelsim_ini "FALSE" \
  -auto_update_viewdraw_ini "FALSE"\
  -block_mode "FALSE" -auto_generate_synth_hdl "TRUE”\
  -auto_file_detection "TRUE"\
  -display_fanout_limit {12}\
  -enable_set_mitigation {1}\
  -enable_design_separation {1}
```

refresh

Tcl command; refreshes your project, updates the view and checks for updated links and files.

```
refresh .
```

Example

```
refresh
```

remove_core

Tcl command; removes a core from your project.

```
remove_core -name core_name
```

Arguments

`-name core_name`

Name of the core you want to remove.

Example

Remove the core ip-beta2:

```
remove_core -name ip-beta2.ccz
```

remove_library

Tcl command; removes a VHDL library from your project.

```
remove_library
-library name
```

Arguments

`-library name`

Specifies the name of the library you wish to remove.

Example

Remove (delete) a library called 'my_lib'.

```
remove_library –library my_lib
```

See Also

add_library

rename_library

remove_profile

Tcl command; deletes a tool profile.

```
remove_profile -name profilename
```

Arguments

`-name profilename`

Specifies the name of the profile you wish to delete.

Example

The following command deletes the profile 'custom1':

```
remove_profile -name custom1
```

rename_file

This Tcl command renames a constraint file specified by the `–file` parameter to a different name specified by the `–target` parameter.

```
rename_file -file {filename} -target {new_filename}
```

Arguments

`-file {filename}`

Specifies the original name of the file.

`-target {new_filename}`

Specifies the new name of the file.

Example

This command renames the file `a.sdc` to `b.sdc`.

```
rename_file -file {c:/user/a.sdc} -target {c:/user/b.sdc}
```

Return Value

This command returns 0 on success and 1 on failure.

rename_library

Tcl command; renames a VHDL library in your project.

```
rename_library
-library name
 -name name
```

Arguments

`-library name`

Identifies the current name of the library that you wish to rename.

`-name name`

Specifies the new name of the library.

Example

Rename a library from 'my_lib' to 'test_lib1'

```
rename_library –library my_lib -name test_lib1
```

See Also

add_library
remove_library

run_tool

run_tool starts the specified tool. For tools that support command files, an optional command file can be supplied through the -script parameter.

```
run_tool
-name {<tool_name >} \
-script {<absolute or relative path to script file>}
```

-script is an optional parameter.

```
tool_name ::= SYNTHESIZE | COMPILE | SIM_PRESYNTH | SIM_POSTSYNTH | SIM_POSTLAYOUT |
PLACEROUTE | VERIFYTIMING | VERIFYPOWER | GENERATEPROGRAMMINGFILE | GENERATE_MEMORY_MAP
| PROGRAMDEVICE | CONFIGURE_CHAIN | SMARTDEBUG | SSNANALYZER  | GENERATE_SPI_FLASH_IMAGE
| PROGRAM_SPI_FLASH_IMAGE
```

Return

run_tool returns 0 on success and 1 on failure.

Supported tool_names

The following table lists tool_names for run_tool –name {*tool_name*}.

tool_name	Parameter	Description
SYNTHESIZE	-script {*script_file*}	Runs synthesis on your design.
COMPILE	N/A	Runs Compile with default or configured settings.
SIM_PRESYNTH	N/A	Runs pre-synthesis simulation with your default simulation tool
SIM_POSTSYNTH	N/A	Runs post-synthesis simulation with your default simulation tool.
SIM_POSTLAYOUT	N/A	Runs post-layout simulation with your default simulation tool.
PLACEROUTE	N/A	Runs Layout with default or configured settings.
VERIFYTIMING	-script {*script_file*}	Runs timing analysis with default settings/configured settings in *script_file*.
VERIFYPOWER	-script {*script_file*}	Runs power analysis with default settings/configured settings in *script_file*.

tool_name	Parameter	Description
GENERATEPROGRAMMINGFILE	N/A	Generates the bitstream used for programming within Libero.
GENERATE_MEMORY_MAP	N/A	Exports an XML file in <prj_folder> component/work/<design> /<design>_DataSheet.xml. The file contains information about your root SmartDesign in your project.
PROGRAMDEVICE	N/A	Programs your device with configured parameters.
CONFIGURE_CHAIN	-script {*script_file*}	Takes a script that contains FlashPro-specific Tcl commands and passes them to FlashPro Express for execution.
SMARTDEBUG	-script {*script_file*}	Takes a script that contains SmartDebug-specific Tcl commands and passes them to SmartDebug for execution.
GENERATE_SPI_FLASH_IMAGE	N/A	Generates SPI Flash Image file used for programming SPI FLASH Image within Libero.
PROGRAM_SPI_FLASH_IMAGE	N/A	Programs SPI Flash Image with configured parameters.

`-script {`*`absolute or relative path to script file`*`}`

Script file location.

Example

```
run_tool \
  -name {COMPILE}
run_tool \
  -name {SYNTHESIZE} -script {./control_synopsys.tcl}
  #control _synopsys.tcl contains the synthesis-specific Tcl commands
run_tool \
  -name {VERIFYTIMING} \
  -script {./SmartTime.tcl}
  # Script file contains SmartTime-specific Tcl commands
run_tool \
  -name {VERIFYPOWER} \
  -script {./SmartPower.tcl}
  # Script file contains SmartPower-specific Tcl commands
run_tool \
  -name {SMARTDEBUG}
  -script {./sd_test.tcl}
  # Script file contains SmartDebug-specific Tcl commands
```

Note

Where possible, the value of *tool_name* corresponds to the name of the tool in Libero SoC.

Invoking some tools will cause Libero SoC to automatically run some upstream tools in the design flow. For example, invoking Place and Route will invoke Synthesis (if not already run) before it runs Place and Route.

save_project_as

Tcl command; the save_project_as command saves the current project in Libero SoC with a different name and in a specified directory. You must specify a location with the -location parameter.

```
save_project_as
-name project_name
-location project_location
-files value
-designer_views value
-replace_links value
```

Arguments

`-name project_name`

Specifies the name of your new project.

`-location project_location`

Must include the complete path of the PRJ file. If you do not provide the full path, Libero SoC infers that you want to save the project to your current working directory. This is a required parameter.

`-files value`

Specifies the files you want to copy into your new project.

Value	Description
all	Copies all your files into your new project
project	Copies only your Libero SoC project files into your new project
source	Copies only the source files into your new project
none	Copies none of the files into your new project; useful if you wish to manually copy only specific project files

`-designer_views value`

Specifies the Designer views you wish to copy into your new project.

Value	Description
all	Copies all your Designer views into your new project
current	Copies only your current Designer fiew files into your new project
none	Copies none of your views into your new project

`-replace_links value`

Specifies whether or not you want to update your file links in your new project.

Value	Description
true	Replaces (updates) the file links in your project during your save
false	Saves your project without updating the file links

Example

Saves your current Libero SoC project as mydesign.prj in the c:/netlists/testprj/mydesign directory:

```
save_project_as -location c:/netlists/testprj/mydesign -name mydesign.prj
```

See Also

new_project
open_project
save_project

save_log

Tcl command; saves your Libero SoC log file.

```
save_log -file value
```

Arguments

`-file` *`value`*

Value is your name for the new log file.

Example

Save the log file file_log.

```
save_log -file file_log
```

See Also

close_project
new_project

save_project

Tcl command; the save_project command saves the current project in Libero SoC.

```
save_project
```

Arguments

None

Example

Saves the project in your current working directory:

```
save_project
```

See Also

new_project
open_project

select_profile

Tcl command; selects a profile to use in your project.

```
select_profile -name profilename
```

Arguments

`-name profilename`

Specifies the name of the profile you wish to use.

Example

The following command selects the profile 'custom1':

```
select_profile -name custom1
```

set_actel_lib_options

Tcl command; the set_actel_lib_options command sets your simulation library to default, or to another library (when you specify a path.

```
set_actel_lib_options -use_default_sim_path value -sim_path {path}
```

Arguments

`-use_default_sim_path value`

Possible values are:

Value	Description
TRUE	Uses the default simulation library.
FALSE	Disables the default simulation library; enables you to specify a different simulation library with the -sim_path {path} option.

`-sim_path {path}`

Specifies the path to your simulation library.

Example

Uses a simulation library in the directory c:\sim_lib\test.

```
set_actel_lib_options -use_default_sim_path FALSE -sim_path {c:\sim_lib\test}
```

set_as_target

This Tcl command sets a SDC, PDC or FDC file as the target file to receive and store new constraints.

```
set_as_target -type {constraint_file_type} \
-file {constraint_file_path}
```

Arguments

`-type {sdc | pdc | fdc}`

Specifies the file type: SDC, PDC, or FDC.

Example

This command sets the SDC file <project_folder> /constraints/user.sdc as the target to receive and store new SDC commands.

```
set_as_target -type {sdc} -file {./constraint/user.sdc}
```

This command sets the PDC file <project_folder> /constraints/user.pdc as the target to receive and store new PDC commands.

```
set_as_target -type {pdc} -file {./constraint/user.pdc}
```

Return Value

This command returns 0 on success and 1 on failure.

set_device (Project Manager)

Tcl command; sets your device family, die, and package in the Project Manager.

```
set_device [-family family] [-die die] [-package package].[-speed speed_grade] [-adv_options
value]
```

Arguments

`-family family`

Sets device family.

`-die die`

Sets device die.

`-package package`

Sets device package.

`-speed speed_grade`

Sets device speed grade.

`-adv_options value`

Sets your advanced options, such as temperature and voltage settings.

Value	Description
IO_DEFT_STD:LVTTL	Sets your I/O default value to LVTTL
TEMPR:COM	Sets your default temperature range; can be COM (Commercial), MIL (Military) or IND (industrial).
VCCI_1.5_VOLTR:COM	Sets VCCI to 1.5 and voltage range to Commercial
VCCI_1.8_VOLTR:COM	Sets VCCI to 1.8 and voltage range to Commercial
VCCI_2.5_VOLTR:COM	Sets VCCI to 2.5 and voltage range to Commercial
VCCI_3.3_VOLTR:COM	Sets VCCI to 3.3 and voltage range to Commercial
VOLTR:COM	Sets your voltage range; can be COM (Commercial), MIL (Military) or IND (industrial).

set_modelsim_options

Tcl command; sets your Model*Sim* simulation options.

```
set_modelsim_options
[-use_automatic_do_file value]
[-user_do_file {path}]
[-sim_runtime {value}]
[-tb_module_name {value}]
[-tb_top_level_name {value}]
[-include_do_file value
[-included_do_file {value}]
[-type {value}]
[-resolution {value}]
[-add_vsim_options {value}]
[-display_dut_wave value]
[-log_all_signals value]
[-do_file_args value]
[-dump_vcd "TRUE | FALSE"]
[-vcd_file "VCD file name"]
```

Arguments

`-use_automatic_do_file value`

Uses an automatic.do file in your project. Possible values are:

Value	Description
TRUE	Uses the default automatic.do file in your project.
FALSE	Uses a different *.do file; use the other simulation options to specify it.

`-user_do_file {path}`

Specifies the location of your user-defined *.do file.

`-sim_runtime {value}`

Sets your simulation runtime. Value is the number and unit of time, such as {1000ns}.

`-tb_module_name {value}`

Specifies your testbench module name, where value is the name.

`-tb_top_level_name {value}`

Sets the top-level instance name in the testbench, where value is the name.

`-include_do_file value`

Includes a *.do file; possible values are:

Value	Description
TRUE	Includes the *.do file.
FALSE	Does not include the *.do file

`-included_do_file {value}`

Specifies the path of the included *.do file, where value is the name of the file.

`-type {value}`

Resolution type; possible values are:

Value	Description
min	Minimum

Value	Description
typ	Typical
max	Maximum

-resolution {*value*}

Sets your resolution value, such as {1ps}.

`-add_vsim_options {value}`

Adds more Vsim options, where value specifies the option(s).

-display_dut_wave *value*

Enables ModelSim to display signals for the tested design; possible values are:

Value	Description
0	Displays the signal for the top_level_testbench
1	Enables ModelSim to display the signals for the tested design

-log_all_signals *value*

Enables you to log all your signals during simulation; possible values are:

Value	Description
TRUE	Logs all signals
FALSE	Does not log all signals

`-do_file_args value`

Specifies *.do file command parameters.

`-dump_vcd value`

Dumps the VCD file when simulation is complete; possible values are:

Value	Description
TRUE	Dumps the VCD file
FALSE	Does not dump the VCD file

`-vcd_file {value}`

Specifies the name of the dumped VCD file, where value is the name of the file.

Example

Sets ModelSim options to use the automatic *.do file, sets simulation runtime to 1000ns, sets the testbench module name to "testbench", sets the testbench top level to <top>_0, sets simulation type to "max", resolution to 1ps, adds no vsim options, does not log signals, adds no additional DO file arguments, dumps the VCD file with a name power.vcd.

```
set_modelsim_options -use_automatic_do_file 1 -sim_runtime {1000ns} -tb_module_name
{testbench} -tb_top_level_name {<top>_0} -include_do_file 0 -type {max} -resolution
{1ps} -add_vsim_options {} -display_dut_wave 0 -log_all_signals 0 -do_file_args {} -
dump_vcd 0 -vcd_file {power.vcd}
```

set_option

Tcl command; sets your synthesis and FPGA Hardware Breakpoint Auto Instantiation options on a module.

```
set_option [-synth "TRUE | FALSE"] [-fhb "TRUE | FALSE"] [-module "module_name"]
```

Arguments

`-synth "TRUE | FALSE"`

Runs synthesis (for a value of TRUE).

`-fhb "TRUE | FALSE"`

Enable/disable FPGA Hardware Breakpoint Auto Instantiation.

`-module module_name`

Identifies the module on which you will run synthesis.

Example

Run synthesis on the module test1.vhd:

```
set_option [-synth TRUE] [-module <module_name>]
```

set_root

Tcl command; sets the module you specify as the root.

```
set_root module_name
```

Arguments

`set_root module_name`

Specifies the name the module you want to set as root.

Example

Set the module mux8 as root:

```
set_root mux8
```

set_user_lib_options

Tcl command; sets your user library options during simulation. If you do not use a custom library these options are not available.

```
set_user_lib_options
-name {value}
-path {path}
-option {value}
```

Arguments

`-name {value}`

Sets the name of your user library.

`-path {path}`

Sets the pathname of your user library.

`-option {value}`

Sets your default compile options on your user library; possible values are:

Value	Description
do_not_compile	User library is not compiled
refresh	User library is refreshed
compile	User library is compiled
recompile	User library is recompiled
refresh_and_compile	User library is refreshed and compiled

Example

The example below sets the name for the user library to "test1", the path to c:/msemi_des_files/libraries/test1, and the compile option to "do not compile".

```
set_user_lib_options -name {test1} -path {c:/msemi_des_files/libraries/test1} -option
{do_not_compile}
```

unlink

Tcl command; removes a link to a file in your project.

```
unlink -file filename [-local local_filename]
```

Arguments

`-file filename`

Name of the linked (remote) file you want to unlink.

`-local local_filename`

Name of the local file that you want to unlink.

Example

Unlink the file hdl1.vhd from my local file test.vhd

```
unlink -file hdl1.vhd [-local test.vhd]
```

unset_as_target

This Tcl command unsets a target file in the Constraints view.

```
unset_as_target -file {filename}
```

Arguments

`-file {filename}`

Specifies the name of the file to be unset as a target.

Example

This command unsets the PDC file <project_folder> /constraints/user.pdc:

```
unset_as_target -file {c:/user/a_io.pdc}
```

Return Value

This command returns 0 on success and 1 on failure.

use_source_file

Tcl command; defines a module for your project.

```
use_source_file
-file value
 -module value
```

Arguments

`-file value`

Specifies the Verilog or VHDL file. Value is the name of the file you wish use (including the full pathname).

`-module value`

Specifies the module in which you want to use the file.

Example

Specify file1.vhd in the ./project/hdl directory, in the module named top.

```
use_source_file –file “./project/hdl/file1.vhd” –module “top"
```

See Also

use_file

SmartDebug Tcl Commands

SmartDebug Tcl Support

The following table lists the Tcl commands related to SmartDebug for PolarFire. Click the command to view more information.

Table 1 · SmartDebug Tcl Commands

Command	Action
Probe	
add_probe_insertion_point	Adds probe points to be connected to user-specified I/Os for probe insertion flow.
add_to_probe_group	Adds the specified probe points to the specified probe group
create_probe_group	Creates a new probe group.
delete_active_probe	Deletes either all or the selected active probes.
load_active_probe_list	Loads the list of probes from the file.
move_to_probe_group	Moves the specified probe points to the specified probe group.
program_probe_insertion	Runs the probe insertion flow on the selected nets.
remove_probe_insertion_point	Deletes an added probe from the probe insertion UI.
set_live_probe	Set Live probe channels A and/or B to the specified probe point (or points).
select_active_probe	Manages the current selection of active probe points to be used by active probe READ operations.
read_active_probe	Reads active probe values from the device.
remove_from_probe_group	Move out the specified probe points from the group.
save_active_probe_list	Saves the list of active probes to a file.
select_active_probe	Manages the current selection of active probe points to be used by active probe READ operations.
ungroup	Disassociates the probes as group.
unset_live_probe	Discontinues the debug function and clears live probe channels.
write_active_probe	Sets the target probe point on the device to the specified value.
LSRAM	
read_lsram	Reads a specified block of large SRAM from the device.

Command	Action
Probe	
write_lsram	Writes a seven bit word into the specified large SRAM location.
uSRAM	
read_usram	Reads a uSRAM block from the device.
write_usram	Writes a seven bit word into the specified uSRAM location.
Transceiver	
loopback_mode	Applies loopback to a specified lane.
smartbert_test	Starts and stops a Smart BERT test and resets error counter.
static_pattern_transmit	Starts and stops a Static Pattern Transmit.
Additional Commands	
get_programmer_info	Lists the IDs of all FlashPRO programmers connected to the machine.
uprom_read_memory	Reads uPROM memory block from the device.

add_probe_insertion_point

This Tcl command adds probe points to be connected to user-specified I/Os for probe insertion flow.

```
add_probe_insertion_point –net net_name -driver driver -pin package_pin_name -port port name
```

Arguments

`-net` *`net_name`*

Name of the net used for probe insertion.

`-driver` *`driver`*

Driver of the net.

`-pin` *`package_pin_name`*

Package pin name (i.e. I/O to which the net will be routed during probe insertion).

`-port` *`port_name`*

User-specified name for the probe insertion point.

Example

```
add_probe_insertion_point -net {count_out_c[0]} -driver {Counter_8bit_0_count_out[0]:Q} -
pin {H5} -port {Probe_Insert0}
```

add_to_probe_group

Tcl command; adds the specified probe points to the specified probe group.

```
add_to_probe_group -name probe_name -group group_name
```

Arguments

`-name` *`probe_name`*

Specifies one or more probes to add.

`-group` *`group_name`*

Specifies name of the probe group.

Example

```
add_to_probe_group -name out[5]:out[5]:Q \
                  -name grp1.out[3]:out[3]:Q \
                  -name out.out[1].out[1]:Q \
                  -group my_new_grp
```

create_probe_group

Tcl command; creates a new probe group.

```
create_probe_group -name group_name
```

Arguments

`-name` *`group_name`*

Specifies the name of the new probe group.

Example

```
create_probe_group -name my_new_grp
```

delete_active_probe

Tcl command; deletes either all or the selected active probes.

Note: You cannot delete an individual probe from the Probe Bus.

```
delete_active_probe -all | -name probe_name
```

Arguments

`-all`

Deletes all active probe names.

`-name` *`probe_name`*

Deletes the selected probe names.

Example

```
delete –all      <- deletes all active probe names
delete -name out[5]:out[5]:Q \
       -name my_grp1.out[1]:out[1]:Q            #deletes the selected probe names
delete -name my_grp1 \
       -name my_bus             #deletes the group, bus and their members.
```

get_programmer_info

This Tcl command lists the IDs of all FlashPRO programmers connected to the machine.

```
get_programmer_info
```

This command takes no arguments.

Example

```
set a [get_programmer_info]
```

load_active_probe_list

Tcl command; loads the list of probes from the file.

```
load_active_probe_list –file file_path
```

Arguments

`-file` *`file_path`*

The input file location.

Example

```
load_active_probe_list -file “./my_probes.txt”
```

loopback_mode

This Tcl command applies loopback to a specified lane.

```
loopback_mode -lane {Physical_Location} -apply -type {loopback_type}
```

Arguments

`-lane {`*`Physical_Location`*`}`

Specify the physical location of the lane.

`-apply`

Apply specified loopback to specified lane.

`-type {`*`loopback_type`*`}`

Specify the loopback type to apply.

Examples

```
loopback_mode -lane {Q3_LANE2} -apply -type {EQ-NearEnd}
loopback_mode -lane {Q3_LANE0} -apply -type {EQ-FarEnd}
loopback_mode -lane {Q0_LANE0} -apply -type {CDRFarEnd}
loopback_mode -lane {Q0_LANE1} -apply -type {NoLpbk}
loopback_mode -lane {Q1_LANE2} -apply -type {EQ-FarEnd}
loopback_mode -lane {Q1_LANE0} -apply -type {NoLpbk}
loopback_mode -lane {Q2_LANE2} -apply -type {EQ-NearEnd}
loopback_mode -lane {Q2_LANE3} -apply -type {CDRFarEnd}
```

move_to_probe_group

Tcl command; moves the specified probe points to the specified probe group.

Note: Probe points related to a bus cannot be moved to another group.

```
move_to_probe_group -name probe_name -group group_name
```

Arguments

```
-name probe_name
```

Specifies one or more probes to move.

```
-group group_name
```

Specifies name of the probe group.

Example

```
move_to_probe_group -name out[5]:out[5]:Q \
                   -name grp1.out[3]:out[3]:Q \
                   -group my_grp2
```

program_probe_insertion

This Tcl command runs the probe insertion flow on the selected nets.

```
program_probe_insertion
```

This command takes no arguments.

read_active_probe

Tcl command; reads active probe values from the device. The target probe points are selected by the select_active_probe command.

```
read_active_probe [-deviceName device_name] [-name probe_name] [-group_name
bus_name|group_name] [-value_type b|h][-file file_path]
```

Arguments

```
-deviceName device_name
```

Parameter is optional if only one device is available in the current configuration.

```
-name probe_name
```

Instead of all probes, read only the probes specified. The probe name should be prefixed with bus or group name if the probe is in the bus or group.

```
-group_name bus_name | group_name
```

Instead of all probes, reads only the specified buses or groups specified here.

```
-value_type b | h
```

Optional parameter, used when the read value is stored into a variable as a string.

b = binary

h = hex

```
-file file_path
```

Optional. If specified, redirects output with probe point values read from the device to the specified file.

Note: When the user tries to read at least one signal from the bus/group, the complete bus or group is read. The user is presented with the latest value for all the signals in the bus/group.

Example

`read_active_probe -group_name {bus1}`

`read_active_probe -group_name {group1}`

To save into variable:

`set a [read_active_probe -group_name {bus_name} -value_type h]` #save read data in hex string

If read values are stored into a variable without specifying value_type parameter, it saves values as a binary string by default.

Example

`set a [read_active_probe ]` #sets variable a as binary string of read values after read_active_probe command.

read_lsram

Tcl command; reads a specified block of large SRAM from the device.

Physical block

```
read_lsram -name block_name -fileName file_name
```

Arguments

`-name` *`block_name`*

Specifies the name for the target block.

`-fileName` *`file_name`*

Optional; specifies the output file name for the data read from the device.

Exceptions

- Array must be programmed and active
- Security locks may disable this function

Example

Reads the LSRAM Block Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K20_IP from the PolarFire device and writes it to the file output.txt.

```
read_lsram -name {Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K20_IP} –
fileName {output.txt}
```

Logical block

```
read_lsram -logicalBlockName block_name -port port_name
```

Arguments

`-logicalBlockName` *`block_name`*

Specifies the name for the user defined memory block.

`-port` *`port_name`*

Specifies the port for the memory block selected. Can be either Port A or Port B.

Example

```
read_lsram -logicalBlockName {Fabric_Logic_0/U2/F_0_F0_U1} -port {Port A}
```

read_usram

Tcl command; reads a uSRAM block from the device.

Physical block

```
read_usram [-name block_name] -fileName file_name
```

Arguments

`-name block_name`

Specifies the name for the target block.

`-fileName file_name`

Optional; specifies the output file name for the data read from the device.

Exceptions

- Array must be programmed and active
- Security locks may disable this function

Example

Reads the uSRAM Block Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP from the PolarFire device and writes it to the file sram_block_output.txt.

```
read_usram -name {Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP} –
fileName {output.txt}
```

Logical block

```
read_usram -logicalBlockName block_name -port port_name
```

Arguments

`-logicalBlockName block_name`

Specifies the name of the user defined memory block.

`-port port_name`

Specifies the port of the memory block selected. Can be either Port A or Port B.

Example

```
read_usram -logicalBlockName {Fabric_Logic_0/U3/F_0_F0_U1} -port {Port A}
```

remove_from_probe_group

Tcl command; removes the specified probe points from the group. That is, the removed probe points won't be associated with any probe group.

Note: Probes cannot be removed from the bus.

```
remove_from_probe_group -name probe_name
```

Arguments

`-name probe_name`

Specifies one or more probe points to remove from the probe group.

Example

The following command removes two probes from my_grp2.

```
Move_out_of_probe_group -name my_grp2.out[3]:out[3]:Q \
-name my_grp2.out[3]:out[3]:Q
```

remove_probe_insertion_point

This Tcl command deletes an added probe from the probe insertion UI.

```
remove_probe_insertion_point –net net_name -driver driver
```

Arguments

`-net net_name`

Name of the existing net which is added using the `add_probe_insertion_point` command.

`-driver driver`

Driver of the net.

Example

```
remove_probe_insertion_point -net {count_out_c[0]} -driver
{Counter_8bit_0_count_out[0]:Q}
```

save_active_probe_list

Tcl command; saves the list of active probes to a file.

```
save_active_probe_list -file file_path
```

Arguments

`-file file_path`

The output file location.

Example

```
save_active_probe_list -file “./my_probes.txt”
```

select_active_probe

Tcl command; manages the current selection of active probe points to be used by active probe READ operations. This command extends or replaces your current selection with the probe points found using the search pattern.

```
select_active_probe [-deviceName device_name] [–name probe_name_pattern] [-reset true|false]
```

Arguments

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration..

`-name probe_name_pattern`

Specifies the name of the probe. Optionally, search pattern string can specify one or multiple probe points. The pattern search characters “*” and “?” also can be specified to filter out the probe names.

-reset `true | false`

Optional parameter; resets all previously selected probe points. If name is not specified, empties out current selection.

Example

The following command selects three probes. In the below example, “grp1” is a group and “out” is a bus..

```
Select_active_probe -name out[5]:out[5]:Q
Select_active_probe -name out.out[1]:out[1]:Q \
                   -name out.out[3]:out[3]:Q \
                   -name out.out[5]:out[5]:Q
```

set_live_probe

Tcl command; set_live_probe channels A and/or B to the specified probe point(s). At least one probe point must be specified. Only exact probe name is allowed (i.e. no search pattern that may return multiple points).

```
set_live_probe [-deviceName device_name] [–probeA probe_name] [–probeB probe_name]
```

Arguments

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug user guide for details).

`-probeA` *`probe_ name`*

Specifies target probe point for the probe channel A.

-probeB *`probe_ name`*

Specifies target probe point for the probe channel B.

Exceptions

- The array must be programmed and active
- Active probe read or write operation will affect current settings of Live probe since they use same probe circuitry inside the device
- Setting only one Live probe channel affects the other one, so if both channels need to be set, they must be set from the same call to set_live_probe
- Security locks may disable this function
- In order to be available for Live probe, ProbeA and ProbeB I/O's must be reserved for Live probe respectively

Example

Sets the Live probe channel A to the probe point A12 on device MPF300TS_ES.

```
set_live_probe [-deviceName MPF300TS_ES] [–probeA A12]
```

smartbert_test

This Tcl command is used for the following:

- Start a Smart BERT test
- Stop a Smart BERT test
- Reset error count

smartbert_test -start

This Tcl command starts a Smart BERT test with a specified pattern on a specified lane.

```
smartbert_test -start -pattern {pattern_type} -lane {Physical_Location}
```

Arguments

`-start`

Start the Smart BERT test.

`pattern {pattern_type}`

Specify the pattern type of the Smart BERT test.

`-lane{Physical_Location}`

Specify the physical location of the lane.

`-EQ-NearEndLoopback`

Enable EQ-Near End Loopback on specified lane.

Examples

```
smartbert_test -start -pattern {prbs9} -lane {Q0_LANE3}
smartbert_test -start -pattern {prbs23} -lane {Q3_LANE2}
smartbert_test -start -pattern {prbs7} -lane {Q3_LANE1}
smartbert_test -start -pattern {prbs31} -lane {Q1_LANE2} -EQ-NearEndLoopback
smartbert_test -start -pattern {prbs9} -lane {Q2_LANE2} -EQ-NearEndLoopback
smartbert_test -start -pattern {prbs15} -lane {Q2_LANE3} -EQ-NearEndLoopback
```

smartbert_test -stop

This Tcl command stops a Smart BERT test on a specified lane.

```
smartbert_test -stop -lane {Physical_Location}
```

Arguments

`-stop`

Stop the smart BERT test.

`-lane {Physical_Location}`

Specify the physical location of the lane.

Examples

```
smartbert_test -stop -lane {Q0_LANE0}
smartbert_test -stop -lane {Q0_LANE3}
smartbert_test -stop -lane {Q3_LANE2}
smartbert_test -stop -lane {Q3_LANE1}
smartbert_test -stop -lane {Q1_LANE2}
smartbert_test -stop -lane {Q2_LANE2}
smartbert_test -stop -lane {Q2_LANE3}
```

smartbert_test -reset_counter

This Tcl command resets a lane error counter.

```
smartbert_test -reset_counter -lane {Physical_Location}
```

Arguments

`-reset_counter`

Reset lane error counter on hardware and cumulative error count on the UI.

`-lane {Physical_Location}`

Specify the physical location of the lane.

Examples

```
smartbert_test -reset_counter -lane {Q0_LANE0}
smartbert_test -reset_counter -lane {Q3_LANE2}
smartbert_test -reset_counter -lane {Q2_LANE3}
smartbert_test -reset_counter -lane {Q2_LANE2}
smartbert_test -reset_counter -lane {Q1_LANE2}
smartbert_test -reset_counter -lane {Q3_LANE1}
```

static_pattern_transmit

This Tcl command starts and stops a Static Pattern Transmit.

static_pattern_transmit -start

```
static_pattern_transmit -start -lane {Physical_Location} -pattern {pattern_type} -value
{user_pattern_value}
```

Parameters

`-start`

Start the Static Pattern Transmit.

`-lane {Physical_Location}`

Specify physical location of lane.

`-pattern {pattern_type}`

Specify pattern_type of Static Pattern Transmit.

`-value {user_pattern_value`

Specify user_pattern_value in hex if pattern_type selected is custom.

Examples

```
static_pattern_transmit -start -lane {Q0_LANE0} -pattern {fixed}
static_pattern_transmit -start -lane {Q0_LANE2} -pattern {maxrunlength} -value {}
static_pattern_transmit -start -lane {Q3_LANE2} -pattern {custom} -value {df}
static_pattern_transmit -start -lane {Q3_LANE0} -pattern {fixed} -value {}
static_pattern_transmit -start -lane {Q1_LANE1} -pattern {custom} -value {4578}
static_pattern_transmit -start -lane {Q1_LANE2} -pattern {fixed} -value {}
static_pattern_transmit -start -lane {Q2_LANE2} -pattern {maxrunlength} -value {}
static_pattern_transmit -start -lane {Q2_LANE1} -pattern {custom} -value {abcdef56}
```

static_pattern_transmit -stop

```
static_pattern_transmit -stop -lane {Physical_Location}
```

Parameters

`-stop`

Stop the Static Pattern Transmit.

`-lane {Physical_Location}`

Specify physical location of lane.

Examples

```
static_pattern_transmit -stop -lane {Q0_LANE0}
static_pattern_transmit -stop -lane {Q0_LANE2}
static_pattern_transmit -stop -lane {Q3_LANE2}
static_pattern_transmit -stop -lane {Q3_LANE0}
static_pattern_transmit -stop -lane {Q1_LANE1}
static_pattern_transmit -stop -lane {Q1_LANE2}
static_pattern_transmit -stop -lane {Q2_LANE2}
static_pattern_transmit -stop -lane {Q2_LANE1}
```

ungroup

Tcl command; disassociates the probes as a group.

```
nngroup -name group_name
```

Arguments

`-name group_name`

Name of the group.

Example

```
ungroup –name my_grp4
```

unset_live_probe

Tcl command; discontinues the debug function and clears live probe A, live probe B, or both probes (Channel A/Channel B). An all zeros value is shown in the oscilloscope.

```
unset_live_probe -probeA 1 -probeB 1 [-deviceName device_name]
```

Arguments

`-probeA`

Live probe Channel A.

`-probeB`

Live probe Channel B.

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration or set for debug (see the SmartDebug User's Guide for details).

Exceptions

- The array must be programmed and active.
- Active probe read or write operation affects current of Live Probe settings, because they use the same probe circuitry inside the device.
- Security locks may disable this function.

Example

The following example unsets live probe Channel A from the device MPF300TS_ES.

```
unset_live_probe -probeA 1[-deviceName MPF300TS_ES]
```

uprom_read_memory

This Tcl command reads a uPROM memory block from the device.

```
read_uprom_memory –startAddress {hex_value} –words {integer_value}
```

Arguments

`-startAddress hex_value`

Specifies the start address of the uPROM memory block.

`-words integer_value`

Specifies the number of 9-bit words.

Example

```
read_uprom_memory –startAddress {0xA} –words {100}
```

write_active_probe

Tcl command; sets the target probe point on the device to the specified value. The target probe point name must be specified.

```
write_active_probe [-deviceName device_name] –name probe_name -value true|false
-group_name group_bus_name -group_value "hex-value" | "binary-value"
```

Arguments

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration.

`-name probe_name`

Specifies the name for the target probe point. Cannot be a search pattern.

`-value true | false hex-value | binary-value`

Specifies values to be written.

True = High

False = Low

`-group_name group_bus_name`

Specify the group or bus name to write to complete group or bus.

`-group_value "hex-value" | "binary-value"`

Specify the value for the complete group or bus.

Hex-value format : " <size>'h<value>"

Binary-value format: " <size>'b<value>"

Example

```
write_active_probe –name out[5]:out[5]:Q –value true <-- write to a single probe
write_active_probe -name grp1.out[3]:out[3]:Q -value low <-- write to a probe in the group
write_active_probe -group_name grp1 –group_value "8'hF0" <-- write the value to complete group
write_active_probe –group_name out –group_value "8'b11110000" \
                    -name out[2]:out[2]:Q –value true <-- write multiple probes at the same time.
```

write_lsram

Tcl command; writes a word into the specified large SRAM location.

Physical block

```
write_lsram –name block_name] –offset offset_value –value integer_value
```

Arguments

`-name` *`block_name`*

Specifies the name for the target block.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-value` *`integer_value`*

Word to be written to the target location. Depending on the configuration of memory blocks, the width can be 1, 2, 5, 10, or 20 bits.

Exceptions

- Array must be programmed and active
- The maximum value that can be written depends on the configuration of memory blocks
- Security locks may disable this function

Example

```
write_lsram -name {Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K20_IP} -offset
0 -value 291
```

Logical block

```
write_lsram -logicalBlockName block_name -port port_name -offset 1 offset_value -logicalValue
hexadecimal_value
```

Arguments

`-logicalBlockName` *`block_name`*

Specifies the name of the user defined memory block.

`-port` *`port_name`*

Specifies the port of the memory block selected. Can be either Port A or Port B.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-logicalValue` *`hexadecimal_value`*

Specifies the hexadecimal value to be written to the memory block. Size of the value is equal to the width of the output port selected.

Example

```
write_lsram -logicalBlockName {Fabric_Logic_0/U2/F_0_F0_U1} -port {Port A} -offset 1 -
logicalValue {00FFF}
```

write_usram

Tcl command; writes a 12-bit word into the specified uSRAM location.

Physical block

```
write_usram –name block_name] –offset offset_value –value integer_value
```

Arguments

`-name` *`block_name`*

Specifies the name for the target block.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-value` *`integer_value`*

12-bit value to be written.

Exceptions

- Array must be programmed and active
- The maximum value that can be written is 0x1FF
- Security locks may disable this function

Example

Writes a value of 0x291 to the device PolarFire in the Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP with an offset of 0.

```
write_lsram -name {Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP} -
offset 0 -value 291
```

Logical block

```
write_usram -logicalBlockName block_name -port port_name -offset offset_value -logicalValue
hexadecimal_value
```

Arguments

`-logicalBlockName` *`block_name`*

Specifies the name of the user defined memory block.

`-port` *`port_name`*

Specifies the port of the memory block selected. Can be either Port A or Port B.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-logicalValue` *`hexadecimal_value`*

Specifies the hexadecimal value to be written to the memory block. Size of the value is equal to the width of the output port selected.

Example

```
write_usram -logicalBlockName {Fabric_Logic_0/U3/F_0_F0_U1} -port {Port A} -offset 1 -
logicalValue {00FFF}
```

SmartPower Tcl Commands

smartpower_add_new_scenario

Tcl command; creates a new scenario.

```
smartpower_add_new_scenario -name {value} -description {value} -mode {value}
```

Arguments

`-name {value}`

Specifies the name of the new scenario.

`-description {value}`

Specifies the description of the new scenario.

`-mode {<operating mode>:<duration>}+`

Specifies the mode(s) and duration(s) for the specified scenario.

Examples

This example creates a new scenario called myscenario:

```
smartpower_add_new_scenario -name "MyScenario" -mode "Custom_1:50.00"
"Custom_2:25.00" -mode "Active:25.00"
```

See Also

Tcl documentation conventions

Tcl command; adds a pin into a clock or set domain.

```
smartpower_add_pin_in_domain -pin_name {pin_name} -pin_type {value} –domain_name
{domain_name} -domain_type {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin to add to the domain.

`-pin_type {value}`

Specifies the type of the pin to add. The following table shows the acceptable values for this argument:

Value	Description
clock	The pin to add is a clock pin
data	The pin to add is a data pin

`-domain_name {domain_name}`

Specifies the name of the domain in which to add the specified pin.

`-domain_type {value}`

Specifies the type of domain in which to add the specified pin. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

Notes

- The `domain_name` must be a name of an existing domain.
- The `pin_name` must be a name of a pin that exists in the design.

Examples

The following example adds a clock pin to an existing Clock domain:

```
smartpower_add_pin_in_domain -pin_name { XCMP3/U0/U1:Y } -pin_type {clock} –domain_name
{clk1} -domain_type {clock}
```

The following example adds a data pin to an existing Set domain:

```
smartpower_add_pin_in_domain -pin_name {XCMP3/U0/U1:Y} -pin_type {data} -domain_name
{myset} -domain_type {set}
```

See Also

Tcl documentation conventions

`smartpower_remove_pin_of_domain`

smartpower_battery_settings

This SmartPower Tcl command sets the battery capacity in SmartPower. The battery capacity is used to compute the battery life of your design.

```
smartpower_battery_settings -capacity {decimal value}
```

Parameters

`-capacity {decimal value}`

Value must be a positive decimal.

This parameter is mandatory.

Exceptions

None

Returns

This command does not return a value.

Usage

The following table lists the parameters for the command, their types, and the values they can be set to.

smartpower_battery_settings	Type	Value	Description
capacity	Decimal	Positive decimal	Specify the battery capacity in mA*Hours

Example

This example sets the battery capacity to 1800 mA * Hours.

```
smartpower_battery_settings –capacity {1800}
```

smartpower_change_clock_statistics

Tcl command; changes the default frequencies and probabilities for a specific domain.

```
smartpower_change_clock_statistics -domain_name {value} -clocks_freq {value} -
clocks_proba {value} -registers_freq {value} -registers_proba {value} -set_reset_freq
{value} -set_reset_proba {value} -primaryinputs_freq {value} -primaryinputs_proba {value} -
combinational_freq {value} -combinational_proba {value}
```

Arguments

`-domain_name{value}`

Specifies the domain name in which to initialize frequencies and probabilities.

`-clocks_freq {value}`

Specifies the user input frequency in Hz, KHz, or MHz for all clocks.

`-clocks_proba {value}`

Specifies the user input probability in % for all clocks.

`-registers_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-registers_proba {value}`

Specifies the user input probability in % for all registers.

`-set_reset_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-set_reset_proba {value}`

Specifies the user input probability in % for all set/reset nets.

`-primaryinputs_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-primaryinputs_proba {value}`

Specifies the user input probability in % for all primary inputs.

`-combinational_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-combinational_proba {value}`

Specifies the user input probability in % for all combinational combinational output.

Note: This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all clocks with:

```
smartpower_change_clock_statistics -domain_name {my_domain} -clocks_freq {10 MHz} -
clocks_proba {20} -registers_freq {10 MHz} -registers_proba {20} -set_reset_freq {10
MHz} -set_reset_proba {20} -primaryinputs_freq {10 MHz} -primaryinputs_proba {20} -
combinational_freq {10 MHz} -combinational_proba {20}
```

See Also

Tcl documentation conventions

smartpower_change_setofpin_statistics

Tcl command; changes the default frequencies and probabilities for a specific set.

```
smartpower_change_setofpin_statistics -domain_name {value} -data_freq {value} -
data_proba {value}
```

Arguments

`-domain_name{value}`

Specifies the domain name in which to initialize data frequencies and probabilities.

`-data_freq {value}`

Specifies the user input data frequency in Hz, KHz, or MHz for all sets of pins.

`-data_proba {value}`

Specifies the user input data probability in % for all sets of pins.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all clocks withs:

```
smartpower_change_setofpin_statistics -domain_name {my_domain} -data_freq {10 MHz} -
data_proba {20}
```

See Also

Tcl documentation conventions

smartpower_commit

Tcl command; saves the changes to the design file.

```
smartpower_commit
```

Arguments

None

Examples

```
smartpower_commit
```

See Also

Tcl documentation conventions

smartpower_compute_vectorless

This Tcl command executes a vectorless analysis of the current operating mode.

Arguments

None

Example

```
smartpower_compute_vectorless
```

See Also

Tcl Command Documentation Conventions

smartpower_create_domain

Tcl command; creates a new clock or set domain.

```
smartpower_create_domain -domain_type {value} -domain_name {domain_name}
```

Arguments

`-domain_type {value}`

Specifies the type of domain to create. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

`-domain_name {domain_name}`

Specifies the name of the new domain.

Notes

The `domain name` cannot be the name of an existing domain.

The domain `type` must be either clock or set.

Examples

The following example creates a new clock domain named "clk2":

```
smartpower_create_domain -domain_type {clock} -domain_name {clk2}
```

The following example creates a new set domain named "myset":

```
smartpower_create_domain -domain_type {set} -domain_name {myset}
```

See Also

Tcl documentation conventions

`smartpower_remove_domain`

smartpower_edit_scenario

Tcl command; edits a scenario.

```
smartpower_edit_scenario -name {value} -description {value} -mode {value} -new_name {value}
```

Arguments

`-name {value}`

Specifies the name of the scenario.

`-description {value}`

Specifies the description of the scenario.

`-mode {<operating mode>:<duration>}`

Specifies the mode(s) and duration(s) for the specified scenario.

`-new_name {value}`

Specifies the new name for the scenario

Examples

This example edits the name of myscenario to finalscenario:

```
smartpower_edit_scenario -name myscenario -new_name finalscenario
```

See Also

Tcl documentation conventions

smartpower_import_vcd

This SmartPower Tcl command imports into SmartPower a VCD file generated by a simulation tool. SmartPower extracts the frequency and probability information from the VCD.

```
import_vcd -file "VCD file" [-opmode "mode name"] [-with_vectorless "TRUE | FALSE"] [-
partial_parse\ "TRUE | FALSE"] [-start_time "decimal value"] [-end_time "decimal value"]
\
[-auto_detect_top_level_name "TRUE | FALSE"] [-top_level_name "top level name"] [-
glitch_filtering\ "false | auto | true"] [-glitch_threshold "integer value"] [-stop_time
"decimal value"]
```

Parameters

`-file "VCD file"`

Value must be a file path. This parameter is mandatory.

`[-opmode "mode name"]`

Value must be a string. This parameter is optional.

`[-with_vectorless "TRUE | FALSE"]`

Value must be a boolean. This parameter is optional.

`[-partial_parse "TRUE | FALSE"]`

Value must be a boolean. This parameter is optional.

`[-start_time "decimal value"]`

Value must be a positive decimal. This parameter is optional.

`[-end_time "decimal value"]`

Value must be a positive decimal. This parameter is optional.

`[-auto_detect_top_level_name "TRUE | FALSE"]`

Value must be a boolean. This parameter is optional.

`[-top_level_name "top level name"]`

Value must be a string. This parameter is optional.

`[-glitch_filtering "false | auto | true"]`

Value must be one of false | auto | true. This parameter is optional.

`[-glitch_threshold "integer value"]`

Value must be a positive integer. This parameter is optional.

Exceptions

None

Returns

This command does not return a value.

Usage

This section lists all the parameters for the command, their types, and the values they can be set to. The default value is always listed first.

smartpower_import_vcd	Type	Values	Description
file	String	Path to a VCD file	Path to a VCD file.
opmode	String	Operating mode name “Active” by default	Operating mode in which the VCD will be imported. If the mode doesn’t exist, it will be created.
with_vectorless	Boolean	TRUE|FALSE	Specify the method to set the frequency and probability information for signals not annotated by the VCD TRUE: use the vectorless analysis FALSE: use average value computed from the VCD.
partial_parse	Boolean	FALSE|TRUE	Enable partial parsing of the VCD. Start time and end time need to be specified when TRUE.
start_time	Decimal value	positive decimal nanoseconds (ns)	Specify the starting timestamp of the VCD extraction in ns. It must be lower than the specified end_time. It must be lower than the last timestamp in the VCD file.
end_time	Decimal value	positive decimal nanoseconds (ns)	Specify the end timestamp of the VCD extraction in ns. It must be higher than the specified start_time.
auto_detect_top_level_name	Boolean	TRUE|FALSE	Enable the auto detection of the top level name in the VCD file. Top_level_name needs to be specified when FALSE.
top_level_name	Boolean	Full hierarchical name	Specify the full hierarchical name of the instance of the design in the VCD file.
glitch_filtering	Boolean	Auto|FALSE|TRUE	AUTO: Enable glitch filtering with predefined thereshold based on the family

smartpower_import_vcd	Type	Values	Description
			TRUE: Enable glitch filtering, glitch_threshold must be specified FALSE: Disable glitch filtering.
glitch_threshold	Integer	Positive integer	Specify the threshold in ps below which glitches are filtered out.

Examples

The Tcl command below imports the power.vcd file generated by the simulator into SmartPower:

```
smartpower_import_vcd –file “../../simulation/power.vcd”
```

The Tcl command below extracts information between 1ms and 2ms in the simulation, and stores the information into a custom mode:

```
smartpower_import_vcd –file “../../simulation/power.vcd” –partial_parse TRUE –start_time
1000000 –end_time 2000000 –opmode “power_1ms_to_2ms”
```

smartpower_init_do

Tcl command; initializes the frequencies and probabilities for clocks, registers, set/reset nets, primary inputs, combinational outputs, enables and other sets of pins, and selects a mode for initialization.

```
smartpower_init_do -with {value} -opmode {value} -clocks {value} -registers {value} -
set_reset {value} -primaryinputs {value} -combinational {value} -enables {value} -othersets
{value}
```

Arguments

`-with{value}`

This sets the option of initializing frequencies and probabilities with vectorless analysis or with fixed values. The following table shows the acceptable values for this argument:

Value	Description
vectorless	Initializes frequencies and probabilities with vectorless analysis
fixed	Initializes frequencies and probabilities with fixed values

`-opmode {value}`

Optional; specifies the mode in which to initialize frequencies and probabilities. The value must be Active or Flash*Freeze.

`-clocks {value}`

This sets the option of initializing frequencies and probabilities for all clocks. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all clocks
false	Does not initialize frequencies and probabilities for all clocks

`-registers {value}`

This sets the option of initializing frequencies and probabilities for all registers. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all registers
false	Does not initialize frequencies and probabilities for all registers

`-set_reset {value}`

This sets the option of initializing frequencies and probabilities for all set/reset nets. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all set/reset nets
false	Does not initialize frequencies and probabilities for all set/reset nets

`-primaryinputs{value}`

This sets the option of initializing frequencies and probabilities for all primary inputs. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all primary inputs
false	Does not initialize frequencies and probabilities for all primary inputs

`-combinational {value}`

This sets the option of initializing frequencies and probabilities for all combinational outputs. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all combinational outputs
false	Does not initialize frequencies and probabilities for all combinational outputs

`-enables {value}`

This sets the option of initializing frequencies and probabilities for all enable sets of pins. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all enable sets of pins
false	Does not initialize frequencies and probabilities for all enable sets of pins

`-othersets {value}`

This sets the option of initializing frequencies and probabilities for all other sets of pins. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all other sets of pins
false	Does not initialize frequencies and probabilities for all other sets of pins

Note: This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all clocks with:

```
smartpower_init_do -with {vectorless} -opmode {my_mode} -clocks {true} -registers {true}
-asynchronous {true} -primaryinputs {true} -combinational {true} -enables {true} -
othersets {true}
```

See Also

Tcl documentation conventions

smartpower_init_set_clocks_options

Tcl command; initializes the clock frequency options of all clock domains.

```
smartpower_init_set_clocks_options -with_clock_constraints {value} -
with_default_values {value} -freq {value} -duty_cycle {value}
```

Arguments

`-with_clock_constraints {value}`

This sets the option of initializing the clock frequencies with frequency constraints from SmartTime. The following table shows the acceptable values for this argument:

Value	Description
true	Sets initialize clock frequencies with clock constraints ON
false	Sets initialize clock frequencies with clock constraints OFF

`-with_default_values {value}`

This sets the option of initializing the clock frequencies with a user input default value. The following table shows the acceptable values for this argument:

Value	Description
true	Sets initialize clock frequencies with default values ON
false	Sets initialize clock frequencies with default values OFF

`-freq {value}`

Specifies the user input frequency in Hz, KHz, or MHz.

`-duty_cycle {value}`

Specifies the user input duty cycles in %.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all clocks after executing `smartpower_init_do` with `-clocks {true}`:

```
smartpower_init_set_clocks_options -with_clock_constraints {true} -with_default_values
{true} -freq {10 MHz} -duty_cycle {20}
```

See Also

Tcl documentation conventions

smartpower_init_set_combinational_options

Tcl commands; initializes the frequency and probability of all combinational outputs.

```
smartpower_init_set_combinational_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all combinational signals after executing `smartpower_init_do` with `-combinational {true}`:

```
smartpower_init_set_combinational_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

smartpower_init_set_enables_options

Tcl command; initializes the clock frequency of all enable clocks with the initialization options.

```
smartpower_init_set_enables_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz).

`-proba {value}`

Specifies the user input probability in %.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all clocks after executing `smartpower_init_do` with `-enables {true}`:

```
smartpower_init_set_enables_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

smartpower_init_set_primaryinputs_options

Tcl command; initializes the frequency and probability of all primary inputs.

```
smartpower_init_set_primaryinputs_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all primary inputs after executing `smartpower_init_do` with `-primaryinputs {true}`:

```
smartpower_init_set_primaryinputs_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

smartpower_init_set_registers_options

Tcl command; initializes the frequency and probability of all register outputs.

```
smartpower_init_set_registers_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all register outputs after executing `smartpower_init_do` with `-registers {true}`:

```
smartpower_init_set_registers_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

smartpower_init_setofpins_values

Tcl command; initializes the frequency and probability of all sets of pins.

```
smartpower_init_setofpins_values -domain_name {name} -freq {value} -proba {value}
```

Arguments

`-domain_name{name}`

Specifies the set of pins that will be initialized. The following table shows the acceptable values for this argument:

Value	Description
IOsEnableSet	Specifies that the IOsEnableSet set of pins will be initialized
MemoriesEnableSet	Specifies that the MemoriesEnableSet set of pins will be initialized

`-freq {value}`

Specifies the user input frequency in Hz, MHz, or KHz.

`-proba {value}`

Specifies the user input probability in %.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all primary inputs after executing `smartpower_init_do` with `-othersets {true}`:

```
smartpower_init_setofpins_values -domain_name {IOsEnableSet} -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

smartpower_remove_all_annotations

Tcl command; removes all initialization annotations for the specified mode.

```
smartpower_remove_all_annotations -opmode {value}
```

Arguments

`-opmode {value}`

Removes all initialization annotations for the specified mode, where value must be Active or Flash*Freeze.

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Examples

The following example initializes all clocks with opmode Acitve:

```
smartpower_remove_all_annotations -opmode {Active}
```

See Also

Tcl documentation conventions

smartpower_remove_file

Tcl command; removes a VCD file from the specified mode or all operating mode. Frequency and probability information of signals annotated by the VCD are set back to the default value.

```
remove_file
-file {value} \
-format {value} \
-opmode {value} \
```

Arguments

`-file {value}`

Specifies the file to be removed. This is mandatory.

`-format VCD`

Specifies that the type to be removed is a VCD file. This is mandatory.

`[-opmode {value}]`

Specifies the operating mode. This is optional. The following table shows the acceptable values for this argument:

Value	Description
Active	The operating mode is set to active
Static	The operating mode is set to Static
Flash*Freeze	The operating mode is set to Flash*Freeze

Examples

This example removes the file test.vcd from the Active mode.

```
smartpower_remove_file -file "test.vcd" -format VCD -opmode "Active"
```

This example removes the VCD file power1.vcd from all operating modes:

```
smartpower_remove_file –file “power1.vcd” –format VCD
```

See Also

Tcl documentation conventions

smartpower_remove_scenario

Tcl command; removes a scenario from the current design.

```
smartpower_remove_scenario -name {value}
```

Arguments

```
-name {value}
```

Specifies the name of the scenario.

Examples

This example removes a scenario from the current design:

```
smartpower_remove_scenario -name myscenario
```

See Also

Tcl documentation conventions

smartpower_report_power

Tcl command; creates a Power report, which enables you to determine if you have any power consumption problems in your design. It includes information about the global device and SmartPower preferences selection, and hierarchical detail (including gates, blocks, and nets), with a block-by-block, gate-by-gate, and net-by-net power summary SmartPower results.

```
smartpower_report_power\
[-powerunit {value}] \
[-frequnit {value}] \
[-opcond {value}] \
[-opmode {value}] \
[-toggle {value}] \
[-power_summary {value}] \
[-rail_breakdown{value}] \
[-type_breakdown{ value}] \
[-clock_breakdown{value}] \
[-thermal_summary {value}] \
[-battery_life {value}] \
[-opcond_summary {value}] \
[-clock_summary {value}] \
[-style {value}] \
[-sortorder {value}] \
[-sortby {value}] \
[-instance_breakdown {value}] \
[-power_threshold {value}] \
[-filter_instance {value}] \
[-min_power {number}] \
[-max_instance {integer >= 0}] \
[-activity_sortorder {value}] \
[-activity_sortby {value}] \
[-activity_summary {value}] \
[-frequency_threshold {value}] \
[-filter_pin {value}] \
[-min_frequency {value}] \
[-max_pin {value}] \
[-enablerates_sortorder {value}] \
[-enablerates_sortby {value}] \
[-enablerates_summary {value}] \
[-with_annotation_coverage {value}] \
{filename}
```

Arguments

`-powerunit {value}`

Specifies the unit in which power is set. The following table shows the acceptable values for this argument:

Value	Description
W	The power unit is set to watts
mW	The power unit is set to milliwatts
uW	The power unit is set to microwatts

`-frequnit {value}`

Specifies the unit in which frequency is set. The following table shows the acceptable values for this argument:

Value	Description
Hz	The frequency unit is set to hertz
kHz	The frequency unit is set to kilohertz
MHz	The frequency unit is set to megahertz

`-opcond {value}`

Specifies the operating condition. The following table shows the acceptable values for this argument:

Value	Description
worst	The operating condition is set to worst case
typical	The operating condition is set to typical case
best	The operating condition is set to best case

`-opmode {value}`

Specifies the operating mode. The following table shows the acceptable values for this argument:

Value	Description
Active	The operating mode is set to Active
Standby	The operating mode is set to Standby
Flash*Freeze	The operating mode is set to Flash*Freeze

`-toggle {value}`

Specifies the toggle. The following table shows the acceptable values for this argument:

Value	Description
true	The toggle is set to true
false	The toggle is set to false

`-power_summary {value}`

Specifies whether to include the power summary, which shows the static and dynamic values in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the power summary in the report
false	Does not include the power summary in the report

`-rail_breakdown {value}`

Specifies whether to include the breakdown by rail summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by rail summary in the report
false	Does not include the breakdown by rail summary in the report

`-type_breakdown {value}`

Specifies whether to include the breakdown by type summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by type summary in the report
false	Does not include the breakdown by type summary in the report

`-clock_breakdown {value}`

Specifies whether to include the breakdown by clock domain in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by clock domain summary in the report
false	Does not include the breakdown by clock domain summary in the report

`-thermal_summary {value}`

Specifies whether to include the thermal summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the thermal summary in the report
false	Does not include the thermal summary in the report

`-battery_life {value}`

Specifies whether to include the battery life summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the battery life summary in the report
false	Does not include the battery life summary in the report

`-opcond_summary {value}`

Specifies whether to include the operating conditions summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the operating conditions summary in the report
false	Does not include the operating conditions summary in the report

`-clock_summary {value}`

Specifies whether to include the clock domains summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the clock summary in the report
false	Does not include the clock summary in the report

`-style {value}`

Specifies the format in which the report will be exported. The following table shows the acceptable values for this argument:

Value	Description
Text	The report will be exported as Text file
CSV	The report will be exported as CSV file

`-sortby {value}`

Specifies how to sort the values in the report. The following table shows the acceptable values for this argument:

Value	Description
power values	Sorts based on the power values
alphabetical	Sorts in an alphabetical order

`-sortorder {value}`

Specifies the sort order of the values in the report. The following table shows the acceptable values for this argument:

Value	Description
ascending	Sorts the values in ascending order
descending	Sorts the values in descending order

`-instance_breakdown {value}`

Specifies whether to include the breakdown by instance in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by instance in the report
false	Does not include the breakdown by instance in the report

`-power_threshold {value}`

This specifies whether to include only the instances that consume power above a certain minimum value. When this command is set to true, the `-min_power` argument must also be used to specify that only the instances that consume power above this minimum power value are the ones that are included in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the power threshold in the report
false	Does not include the power threshold in the report

`-filter_instance {value}`

This specifies whether to have a limit on the number of instances to include in the Power report. When this command is set to true, the `–max_instance` argument must also be used to specify the maximum number of instances to be included into the Power report. The following table shows the acceptable values for this argument:

Value	Description
true	Indicates that you want to have a limit on the number of instances to include in the Power report
false	Indicates that you do not want to have a limit on the number of instances to include in the Power report

`-min_power {number}`

Specifies which block to expand based on the minimum power value of a block.

`-max_instance {integer >= 0}`

Sets the maximum number of instances to a specified integer greater than or equal to 0 (zero). This will limit the maximum number of instances to be included in the Power report.

`-activity_sortorder {value}`

Specifies the sort order for the activity summary. The following table shows the acceptable values for this argument:

Value	Description
ascending	Sorts the values in ascending order
descending	Sorts the values in descending order

`-activity_sortby {value}`

Specifies how to sort the values for the activity summary. The following table shows the acceptable values for this argument:

Value	Description
pin name	Sorts based on the pin name
net name	Sorts based on thepnet name
domain	Sorts based on the clock domain
frequency	Sorts based on the clock frequency
source	Sorts based on the clock frequency source

`-activity_summary {value}`

Specifies whether to include the activity summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the activity summary in the report
false	Does not include the activity summary in the report

`-frequency_threshold {value}`

Specifies whether to add a frequency threshold. The following table shows the acceptable values for this argument:

Value	Description
true	Adds a frequency threshold
false	Does not add a frequency threshold

`-filter_pin {value}`

Specifies whether to filter by maximum number of pins. The following table shows the acceptable values for this argument:

Value	Description
true	Filters by maximum number of pins
false	Des not filter by maximum number of pins

`-min_frequency {value}`

Sets the minimum frequency to {decimal value [unit { Hz | KHz | MHz}]}.

`-max_pin {value}`

Sets the maximum number of pins.

`-enablerates_sortorder {value}`

Specifies the sort order for the probabilities summary. The following table shows the acceptable values for this argument:

Value	Description
ascending	Sorts the values in ascending order
descending	Sorts the values in descending order

`-enablerates_sortby {value}`

Specifies how to sort the values for the probabilities summary. The following table shows the acceptable values for this argument:

Value	Description
pin name	Sorts based on the pin name
net name	Sorts based on the net name
domain	Sorts based on the clock domain
frequency	Sorts based on the clock frequency
source	Sorts based on the clock frequency source

`-enablerates_summary {value}`

Specifies whether to include the probabilities summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the activity summary in the report
false	Does not include the activity summary in the report

`-with_annotation_coverage {value}`

Specifies whether to include the annotation coverage summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the annotation coverage summary in the report
false	Does not include the annotation coverage summary in the report

`{filename}`

Specifies the name of the report.

Supported Families

See the Tcl Commands and Supported Families table for the list of families that support this command.

Notes

- The following arguments have been removed. Running the script will trigger a warning message: Warning: Invalid argument: -argname "argvalue" Ignored. Ignore the warning.

```
-annotated_pins {value}
-stat_pow {value}
-dyn_pow {value}
```

- Flash*Freeze, Sleep, and Shutdown are available only for certain families and devices.
- Worst and Best are available only for certain families and devices.

Examples

This example generates a Power report named report.rpt.

```
smartpower_report_power -powerunit "uW" -frequnit "MHz" -opcond "Typical" -opmode
"Active" -toggle "TRUE" -rail_breakdown "TRUE" -battery_life "TRUE" -style "Text" -
power_summary "TRUE" -activity_sortby "Source" text_report.txt
```

smartpower_set_mode_for_analysis

Tcl command; sets the mode for cycle-accurate power analysis.

```
smartpower_set_mode_for_analysis -mode {value}
```

Arguments

`-mode {value}`

Specifies the mode for cycle-accurate power analysis.

Value	Description
Active	The operating mode is set to Active

Value	Description
Standby	The operating mode is set to Standby
Flash*Freeze	The operating mode is set to Flash*Freeze

Examples

The following example sets the mode for analysis to active:

```
smartpower_set_mode_for_analysis -mode {active}
```

See Also

Tcl documentation conventions

smartpower_set_mode_for_pdpr

This SmartPower Tcl command sets the operating mode used by the Power Driven Place and Route (PDPR) tool during power optimization.

```
smartpower_set_mode_for_pdpr -opmode {value}
```

Parameters

`-opmode {`*`value`*`}`

Value must be a valid operating mode.

This parameter is mandatory.

Sets the operating mode for your power driven place and route.

Exceptions

None

Return Value

This command does not return a value.

Examples

This example sets the Active mode as the operating mode for Power Driven Place and Route.

```
set_mode_for_pdpr -opmode "Active"
```

This example creates a custom mode and set it to be used by Power Driven Place and Route (PDPR).

```
smartpower_add_new_custom_mode –name “MyCustomMode” \
–description “for PDPR” –base_mode “Active”
smartpower_set_mode_for_pdpr –opmode “MyCustomMode
```

See Also

Tcl Command Documentation Conventions

smartpower_set_operating_condition

Tcl command; sets the operating conditions used in SmartPower to one of the pre-defined types.

```
smartpower_set_operating_condition -opcond {value}
```

Arguments

`-opcond {value}`

Specifies the value of the operating condition. The following table shows the acceptable values for this argument:

Value	Description
best	Sets the operating conditions to best
typical	Sets the operating conditions to typical
worst	Sets the operating conditions to worst

Examples

This example sets the operating conditions to best:

```
smartpower_set_operating_condition -opcond {best}
```

See Also

Tcl documentation conventions

smartpower_set_operating_conditions

Tcl command; sets the operating conditions used in SmartPower.

```
smartpower_set_operating_conditions "still_air | 1.0_mps | 2.5_mps | custom" -heatsink
"None | custom | 10mm_Low_Profile | 15mm_Medium_Profile | 20mm_High_Profile" -boardmodel
"None_Conservative | JEDEC_2s2p" [-teta_ja "decimal value"] [-teta_sa "decimal value"]
```

Arguments

`-still_air {value}`

Specifies the value for the still air operating condition. The following table shows the acceptable values for this argument:

Value	Description
1.0_mps	Sets the operating conditions to best
2.5_mps	Sets the operating conditions to typical
custom	Sets the operating conditions to worst

`-heatsink {value}`

Specifies the value of the operating condition. The following table shows the acceptable values for this argument:

Value	Description
none	No heat sink

Value	Description
custom	Sets a custom heat sink size
10mm_Low_Profile	10 mm heat sink
15mm_Low_Profile	15 mm heat sink
20mm_High_Profile	20 mm heat sink

`-boardmodel {value}`

Specifies your board model. The following table shows the acceptable values for this argument:

Value	Description
None_Conservative	No board model, conservative routing
JEDEC_2s2p	JEDEC 2s2p board model

`-teta_ja {decimal_value}`

Optional; sets your teta ja value; must be a positive decimal

`-teta_sa {decimal_value}`

Optional; sets your teta sa value; must be a positive decimal.

Examples

This example sets the operating conditions to best:

```
set_operating_conditions -airflow "still_air" -heatsink "None" -boardmodel
"None_Conservative "
```

See Also

Tcl documentation conventions

smartpower_set_process

Tcl command; sets the process used in SmartPower to one of the pre-defined types.

```
smartpower_set_process -process {value}
```

Arguments

`-process {value}`

Specifies the value of the operating condition. The following table shows the acceptable values for this argument:

Value	Description
Typical	Sets the process for SmartPower to typical
Maximum	Sets the process for SmartPower to maximum

Examples

This example sets the operating conditions to typical:

```
smartpower_set_process -process {Typical}
```

See Also

Tcl documentation conventions

smartpower_set_scenario_for_analysis

Tcl command; sets the scenario for cycle-accurate power analysis.

```
smartpower_set_scenario_for_analysis -scenario{value}
```

Arguments

```
-scenario {value}
```

Specifies the mode for cycle-accurate power analysis.

Examples

The following example sets the scenario for analysis to my_scenario:

```
smartpower_set_scenario_for_analysis -scenario {my_scenario}
```

See Also

Tcl documentation conventions

smartpower_set_temperature_opcond

Tcl command; sets the temperature in the operating conditions to one of the pre-defined types.

```
smartpower_set_temperature_opcond -use{value}
```

Arguments

```
-use{value}
```

Specifies the temperature in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
oprange	Sets the temperature in the operating conditions as specified in your Project Settings.
design	Sets the temperature in the operating conditions as specified in the SmartPower design-wide operating range. Applies to SmartPower only.
mode	Sets the temperature in the operating conditions as specified in the SmartPower mode-specific operating range. Applies to SmartPower only.

Examples

This example sets the temperature in the operating conditions as specified in the custom mode-settings:

```
smartpower_set_temperature_opcond -use{mode}
```

See Also

[Tcl documentation conventions]

smartpower_set_voltage_opcond

Tcl command; sets the voltage in the operating conditions.

```
smartpower_set_voltage_opcond -voltage{value} -use{value}
```

Arguments

`-voltage{value}`

Specifies the voltage supply in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
VDD	Sets the voltage operating conditions for VDD
VDD18	Sets the voltage operating conditions for VDD18
VDDAUX	Sets the voltage operating conditions for VDDAUX
VDDI 1.1	Sets the voltage operating conditions for VDD 1.1
VDDI 1.2	Sets the voltage operating conditions for VDDI 1.2
VDDI 1.35	Sets the voltage operating conditions for VDDI 1.35
VDDI 1.5	Sets the voltage operating conditions for VDDI 1.5
VDDI 1.8	Sets the voltage operating conditions for VDDI 1.8
VDDI 2.5	Sets the voltage operating conditions for VDDI 2.5
VDDI 3.3	Sets the voltage operating conditions for VDDI 3.3
VDD25	Sets the voltage operating conditions for VDD25
VDDA	Sets the voltage operating conditions for VDDA
VDDA25	Sets the voltage operating conditions for VDDA25

`-use{value}`

Specifies the voltage in the operating conditions for each voltage supply. The following table shows the acceptable values for this argument:

Value	Description
oprange	Sets the voltage in the operating conditions as specified in your Project Settings.
design	Sets the voltage in the operating conditions as specified in the SmartPower design-wide operating range. Applies to

Value	Description
	SmartPower only.
mode	Sets the voltage in the operating conditions as specified in the SmartPower mode-specific operating range. Applies to SmartPower only.

Examples

This example sets the VCCA as specified in the SmartPower mode-specific settings:

```
smartpower_set_voltage_opcond -voltage{vcca} -use{mode}
```

See Also

Tcl documentation conventions

smartpower_temperature_opcond_set_design_wide

Tcl command; sets the temperature for SmartPower design-wide operating conditions.

```
smartpower_temperature_opcond_set_design_wide -best{value} -typical{value} -worst{value} -
thermal_mode{value}
```

Arguments

`-best{value}`

Specifies the best temperature (in degrees Celsius) used for design-wide operating conditions.

`-typical{value}`

Specifies the typical temperature (in degrees Celsius) used for design-wide operating conditions.

`-worst{value}`

Specifies the worst temperature (in degrees Celsius) used for design-wide operating conditions.

`-thermal_mode{value}`

Specifies the mode in which the junction temperature is computed. The following table shows the acceptable values for this argument:

Value	Description
ambient	The junction temperature will be iteratively computed with total static power
opcond	The junction temperature will be given as one of the operating condition range values specified in the device selection

Examples

This example sets the temperature for design-wide operating conditions to Best 20, Typical 30, and Worst 60:

```
smartpower_temperature_opcond_set_design_wide -best{20} -typical{30} -worst{60}
```

See Also

Tcl documentation conventions

smartpower_temperature_opcond_set_mode_specific

Tcl command; sets the temperature for SmartPower mode-specific operating conditions.

```
smartpower_temperature_opcond_set_mode_specific -opmode{value} -thermal_mode{value} -
best{value} -typical{value} -worst{value} -thermal_mode{value}
```

Arguments

`-opmode {value}`

Specifies the operating mode. The following table shows the acceptable values for this argument:

Value	Description
Active	The operating mode is set to Active
Static	The operating mode is set to Static
Flash*Freeze	The operating mode is set to Flash*Freeze

`-thermal_mode{value}`

Specifies the mode in which the junction temperature is computed. The following table shows the acceptable values for this argument:

Value	Description
ambient	The junction temperature will be iteratively computed with total static power
opcond	The junction temperature will be given as one of the operating condition range values specified in the device selection

`-best{value}`

Specifies the best temperature (in degrees Celsius) for the selected mode.

`-typical{value}`

Specifies the typical temperature (in degrees Celsius) for the selected mode.

`-worst{value}`

Specifies the worst temperature (in degrees Celsius) for the selected mode.

Examples

This example sets the temperature for mode-specific operating conditions for mode1:

```
smartpower_temperature_opcond_set_mode_specific -mode{mode1} -best{20} -typical{30} -
worst{60}
```

See Also

Tcl documentation conventions

smartpower_voltage_opcond_set_design_wide

Tcl command; sets the voltage settings for SmartPower design-wide operating conditions.

```
smartpower_voltage_opcond_set_design_wide -voltage{value} -best{value} -typical{value} -
worst{value}
```

Arguments

`-voltage{value}`

Specifies the voltage supply in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
VDD	Sets the voltage operating conditions for VDD
VDDI 2.5	Sets the voltage operating conditions for VDDI 2.5
VPP	Sets the voltage operating conditions for VPP
VCCA	Sets the voltage operating conditions for VCCA
VCCI 3.3	Sets the voltage operating conditions for VCCI 3.3
VCCI 2.5	Sets the voltage operating conditions for VCCI 2.5
VCCI 1.8	Sets the voltage operating conditions for VCCI 1.8
VCCI 1.5	Sets the voltage operating conditions for VCCI 1.5
VCC33A	Sets the voltage operating conditions for VCC33A
VCCDA	Sets the voltage operating conditions for VCCDA

`-best{value}`

Specifies the best voltage used for design-wide operating conditions.

`-typical{value}`

Specifies the typical voltage used for design-wide operating conditions.

`-worst{value}`

Specifies the worst voltage used for design-wide operating conditions.

Examples

This example sets VCCA for design-wide to best 20, typical 30 and worst 40:

```
smartpower_voltage_opcond_set_design_wide -voltage{VCCA} -best{20} -typical{30} -
worst{40}
```

See Also

Tcl documentation conventions

smartpower_voltage_opcond_set_mode_specific

Tcl command; sets the voltage settings for SmartPower mode-specific use operating conditions.

```
smartpower_voltage_opcond_set_mode_specific -opmode{value} -voltage{value} -best{value} -
typical{value} -worst{value}
```

Arguments

`-opmode {value}`

Use this option to specify the mode from which the operating conditions are extracted to generate the report.

Value	Description
Active	The operating mode is set to Active
Static	The operating mode is set to Static
Flash*Freeze	The operating mode is set to Flash*Freeze

`-voltage{value}`

Specifies the voltage in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
VDD	Sets the voltage operating conditions for VDD
VDD18	Sets the voltage operating conditions for VDD18
VDDAUX	Sets the voltage operating conditions for VDDAUX
VDDI 1.1	Sets the voltage operating conditions for VDD 1.1
VDDI 1.2	Sets the voltage operating conditions for VDDI 1.2
VDDI 1.35	Sets the voltage operating conditions for VDDI 1.35
VDDI 1.5	Sets the voltage operating conditions for VDDI 1.5
VDDI 1.8	Sets the voltage operating conditions for VDDI 1.8
VDDI 2.5	Sets the voltage operating conditions for VDDI 2.5
VDDI 3.3	Sets the voltage operating conditions for VDDI 3.3
VDD25	Sets the voltage operating conditions for VDD25
VDDA	Sets the voltage operating conditions for VDDA
VDDA25	Sets the voltage operating conditions for VDDA25

`-best{value}`

Specifies the best voltage used for mode-specific operating conditions.

`-typical{value}`

Specifies the typical voltage used for mode-specific operating conditions.

`-worst{value}`

Specifies the worst voltage used for mode-specific operating conditions.

Examples

This example sets the voltage for the static mode and sets best to 20, typical to 30 and worst to 40:

```
smartpower_voltage_opcond_set_mode_specific -opmode{active} -voltage{VCCA} -best{20} -
typical{30} -worst{40}
```

See Also

Tcl documentation conventions

SmartTime Tcl Commands

create_set

Tcl command; creates a set of paths to be analyzed. Use the arguments to specify which paths to include. To create a set that is a subset of a clock domain, specify it with the `-clock` and `-type` arguments. To create a set that is a subset of an inter-clock domain set, specify it with the `-source_clock` and `-sink_clock` arguments. To create a set that is a subset (filter) of an existing named set, specify the set to be filtered with the `-parent_set` argument.

```
create_set\ -name <name>\ -parent_set <name>\ -type <set_type>\ -clock <clock name>\ -
source_clock <clock name>\ -sink_clock <clock name>\ -in_to_out\ -source <port/pin pattern>\
-sink <port/pin pattern>
```

Arguments

`-name <name>`

Specifies a unique name for the newly created path set.

`-parent_set <name>`

Specifies the name of the set to filter from.

`-clock <clock_name>`

Specifies that the set is to be a subset of the given clock domain. This argument is valid only if you also specify the `-type` argument.

`-type <value>`

Specifies the predefined set type on which to base the new path set. You can only use this argument with the `-clock` argument, not by itself.

Value	Description
reg_to_reg	Paths between registers in the design
async_to_reg	Paths from asynchronous pins to registers
reg_to_async	Paths from registers to asynchronous pins
external_recovery	The set of paths from inputs to asynchronous pins
external_removal	The set of paths from inputs to asynchronous pins
external_setup	Paths from input ports to registers
external_hold	Paths from input ports to registers
clock_to_out	Paths from registers to output ports

`-in_to_out`

Specifies that the set is based on the “Input to Output” set, which includes paths that start at input ports and end at output ports.

`-source_clock <clock_name>`

Specifies that the set will be a subset of an inter-clock domain set with the given source clock. You can only use this option with the `-sink_clock` argument.

`-sink_clock <clock_name>`

Specifies that the set will be a subset of an inter-clock domain set with the given sink clock. You can only use this option with the `-source_clock` argument.

`-source <port/pin_pattern>`

Specifies a filter on the source pins of the parent set. If you do not specify a parent set, this option filters all pins in the current design.

`-sink <port/pin_pattern>`

Specifies a filter on the sink pins of the parent set. If you do not specify a parent set, this option filters all pins in the current design.

Examples

```
create_set -name { my_user_set } –source { C* } –sink { D* }
create_set -name { my_other_user_set } –parent_set { my_user_set } –source { CL* }
create_set -name { adder } –source { ALU_CLOCK } –type { REG_TO_REG } -sink { ADDER*}
create_set -name { another_set } –source_clock { EXTERN_CLOCK } –sink_clock {
MY_GEN_CLOCK }
```

expand_path

Tcl command; displays expanded path information (path details) for paths. The paths to be expanded are identified by the parameters required to display these paths with list_paths. For example, to expand the first path listed with list_paths -clock {MYCLOCK} -type {register_to_register}, use the command expand_path -clock {MYCLOCK} -type {register_to_register}. Path details contain the pin name, type, net name, cell name, operation, delay, total delay, and edge as well as the arrival time, required time, and slack. These details are the same as details available in the SmartTime Expanded Path window.

```
expand_path
-index value
-set name
-clock clock name
-type set_type
-analysis {max| min}
-format {csv | text}
-from_clock clock name
-to_clock clock name
```

Arguments

`-index value`

Specify the index of the path to be expanded in the list of paths. Default is 1.

`-analysis {max | min}`

Specify whether the timing analysis is done is max-delay (setup check) or min-delay (hold check). Valid values: max or min.

`-format {csv | text}`

Specify the list format. It can be either text (default) or csv (comma separated values). The former is suited for display the latter for parsing.

`-set name`

Displays a list of paths from the named set. You can either use the -set option to specify a user set by its name or use both -clock and -type to specify a set.

`-clock clock name`

Displays the set of paths belonging to the specified clock domain. You can either use this option along with -type to specify a set or use the -set option to specify the name of the set to display.

`-type set_type`

Specifies the type of paths in the clock domain to display in a list. You can only use this option with the -clock option. You can either use this option along with -clock to specify a set or use the -set option to specify a set name.

Value	Description
reg_to_reg	Paths between registers in the design
external_setup	Path from input ports to registers
external_hold	Path from input ports to registers
clock_to_out	Path from registers to output ports
reg_to_async	Path from registers to asynchronous pins
external_recovery	Set of paths from inputs to asynchronous pins
external_removal	Set of paths from inputs to asynchronous pins
async_to_reg	Path from asynchronous pins to registers

`-from_clock clock_name`

Displays a list of timing paths for an inter-clock domain set belonging to the source clock specified. You can only use this option with the -to_clock option, not by itself.

`-to_clock clock_name`

Displays a list of timing paths for an inter-clock domain set belonging to the sink clock specified. You can only use this option with the -from_clock option, not by itself.

`-analysis name`

Specifies the analysis for the paths to be listed. The following table shows the acceptable values for this argument.

Value	Description
maxdelay	Maximum delay analysis
mindelay	Minimum delay analysis

`-index list_of_indices`

Specifies which paths to display. The index starts at 1 and defaults to 1. Only values lower than the max_paths option will be expanded.

`-format value`

Specifies the file format of the output. The following table shows the acceptable values for this argument:

Value	Description
text	ASCII text format
csv	Comma separated value file format

Examples

Note: The following example returns a list of five paths:

```
puts [expand_path –clock { myclock } –type {reg_to_reg }]
puts [expand_path –clock {myclock} –type {reg_to_reg} –index { 1 2 3 } –format text]
```

See Also

list_paths

list_paths

Tcl command; returns a list of the *n* worst paths matching the arguments. The number of paths returned can be changed using the set_options -limit_max_paths <*value*> command.

```
list_paths
-analysis <max | min>
-format <csv | text>
-set <name>
-clock <clock name>
-type <set_type>
-from_clock <clock name>
-to_clock <clock name>
-in_to_out
-from <port/pin pattern>
-to <port/pin pattern>
```

Arguments

`-analysis <max | min>`

Specifies whether the timing analysis is done for max-delay (setup check) or min-delay (hold check). Valid values are: max or min.

`-format < text | csv >`

Specifies the list format. It can be either text (default) or csv (comma separated values). Text format is better for display and csv format is better for parsing.

`-set <name>`

Returns a list of paths from the named set. You can either use the -set option to specify a user set by its name or use both -clock and -type to specify a set.

`-clock <clock name>`

Returns a list of paths from the specified clock domain. This option requires the -type option.

`-type <set_type>`

Specifies the type of paths to be included. It can only be used along with -clock. Valid values are:

reg_to_reg -- Paths between registers

external_setup -- Path from input ports to data pins of registers

external_hold -- Path from input ports to data pins of registers

clock_to_out -- Path from registers to output ports

reg_to_async -- Path from registers to asynchronous pins of registers

external_recovery -- Path from input ports to asynchronous pins of registers

external_removal -- Path from input ports to asynchronous pins of registers

async_to_reg -- Path from asynchronous pins to registers

`-from_clock <clock name>`

Used along with -to_clock to get the list of paths of the inter-clock domain between the two clocks.

`-to_clock <clock name>`

Used along with -from_clock to get the list of paths of the inter-clock domain between the two clocks.

`-in_to_out`

Used to get the list of path between input and output ports.

`-from <port/pin pattern>`

Filter the list of paths to those starting from ports or pins matching the pattern.

`-to <port/pin pattern>`

Filter the list of paths to those ending at ports or pins matching the pattern.

Example

The following command displays the list of register to register paths of clock domain clk1:

```
puts [ list_paths -clock clk1 -type reg_to_reg ]
```

See Also

create_set

expand_path

set_options

read_sdc

The read_sdc Tcl command evaluate an SDC file, adding all constraints to the specified scenario (or the current/default one if none is specified). Existing constraints are removed if -add is not specified.

```
read_sdc
-add
-scenario scenario_name
-netlist (user | optimized)
-pin_separator (: | /)
file name
```

Arguments

`-add`

Specifies that the constraints from the SDC file will be added on top of the existing ones, overriding them in case of a conflict. If not used, the existing constraints are removed before the SDC file is read.

`-scenario  scenario_name`

Specifies the scenario to add the constraints to. The scenario is created if none exists with this name.

`-netlist (user | optimized)`

Specifies whether the SDC file contains object defined at the post-synthesis netlist (user) level or physical (optimized) netlist (used for timing analysis).

`-pin_separator sep`

Specify the pin separator used in the SDC file. It can be either ':' or '/'.

`file name`

Specify the SDC file name.

Example

The following command removes all constraints from the current/default scenario and adds all constraints from design.sdc file to it:

```
read_sdc design.sdc
```

See Also

write_sdc

remove_set

Tcl command; removes a set of paths from analysis. Only user-created sets can be deleted.

```
remove_set -name name
```

Parameters

```
-name name
```

Specifies the name of the set to delete.

Example

The following command removes the set named my_set:

```
remove_set -name my_set
```

See Also

create_set

report

Tcl command; specifies the type of reports to generate and what to include in the reports.

```
report -type (timing|violations | datasheet|bottleneck | constraints_coverage |
combinational_loops)
       -analysis <max_or_min>\
       -format (csv|text)
        <filename>
        timing options
           -max_parallel_paths <number>
           -max_paths <number>
           -print_summary (yes|no)
           -use_slack_threshold (yes|no)
           -slack_threshold <double>
           -print_paths (yes|no)
           -max_expanded_paths <number>
           -include_user_sets (yes|no)
           -include_clock_domains (yes|no)
           -select_clock_domains <clock name list>
           -limit_max_paths (yes|no)
           -include_pin_to_pin (yes|no)
        bottleneck options
           -cost_type (path_count|path_cost)
           -max_instances <number>
           -from <port/pin pattern>
           -to <port/pin pattern>
           -set_type <set_type>
           -set_name <set name>
           -clock <clock name>
           -from_clock <clock name>
```

```
          -to_clock <clock name>
-in_to_out
```

Arguments

`-type`

Value	Description
timing	Timing Report
violations	Timing Violation Report
constraints_coverage	Constraints Coverage Report
combinational_loops	Combinational Loops Report

`-analysis`

Value	Description
max	Timing report considers maximum analysis (default).
min	Timing report considers minimum analysis.
text	Generates a text report (default).
csv	Generates the report in a comma-separated value format which you can import into a spreadsheet.

`-filename`

Specifies the file name for the generated report.

Timing Options and Values

Parameter/Value	Description
-max_parallel_paths <number>	Specifies the max number of parallel paths. Parallel paths are timing paths with the same start and end points.
-max_paths <number>	Specifies the max number of paths to display for each set. This value is a positive integer value greater than zero. Default is 100.
-print_summary <yes|no>	Yes to include and No to exclude the summary section in the timing report.
-use_slack_threshold <yes|no>	Yes to include slack threshold and no to exclude threshold in the timing report. The default is to exclude slack threshold.
-slack_threshold <double>	Specifies the threshold value to consider when reporting path slacks. This value is in nanoseconds (ns). By default, there is no threshold (all slacks reported).
-print_paths (yes|no)	Specifies whether the path section (clock domains and in-to-out paths) will be printed in the timing report. Yes to include path sections (default) and no to exclude path sections from the timing report.

Parameter/Value	Description
-max_expanded_paths <number>	Specifies the max number of paths to expand per set. This value is a positive integer value greater than zero. Default is 100.
-include_user_sets (yes|no)	If yes, the user set is included in the timing report. If no, the user set is excluded in the timing report.
-include_clock_domains (yes|no)	Yes to include and no to exclude clock domains in the timing report.
-select_clock_domains <clock_name_list>	Defines the clock domain to be considered in the clock domain section. The domain list is a series of strings with domain names separated by spaces. Both the summary and the path sections in the timing report display only the listed clock domains in the clock_name_list.
-limit_max_paths (yes|no)	Yes to limit the number of paths to report. No to specify that there is no limit to the number of paths to report (the default).
-include_pin_to_pin (yes|no)	Yes to include and no to exclude pin-to-pin paths in the timing report.

Bottleneck Options and Values

Parameter/Value	Description
-cost_type <path_count|path_cost>	Specifies the cost_type as either path_count or path_cost. For path_count, instances with the greatest number of path violations will have the highest bottleneck cost. For path_cost, instances with the largest combined timing violations will have the highest bottleneck cost.
-max_instances <number>	Specifies the maximum number of instances to be reported. Default is 10.
-from <port/pin pattern>	Reports only instances that lie on violating paths that start at locations specified by this option.
-to <port/pin pattern>	Reports only instances that lie on violating paths that end at locations specified by this option.
-clock <clock name>	This option allows pruning based on a given clock domain. Only instances that lie on these violating paths are reported.
-set_name <set name>	Displays the bottleneck information for the named set. You can either use this option or use both -clock and -type. This option allows pruning based on a given set. Only paths that lie within the named set will be considered towards bottleneck.
-set_type <set_type>	This option can only be used in combination with the -clock option, and not by itself. The options allows you to filter which type of paths should be considered towards the bottleneck: • reg_to_reg - Paths between registers in the design • async_to_reg - Paths from asynchronous pins to registers • reg_to_async - Paths from registers to asynchronous pins • external_recovery - The set of paths from inputs to

Parameter/Value	Description
	asynchronous pins • external_removal - The set of paths from inputs to asynchronous pins • external_setup - Paths from input ports to registers • external_hold - Paths from input ports to registers • clock_to_out - Paths from registers to output ports
-from_clock <clock name>	Reports only bottleneck instances that lie on violating timing paths of the inter-clock domain that starts at the source clock specified by this option. This option can only be used in combination with -to_clock.
-to_clock <clock name>	Reports only instances that lie on violating paths that end at locations specified by this option.
-in_to_out	Reports only instances that lie on violating paths that begin at input ports and end at output ports.

Example

The following example generates a timing violation report named timing_viol.txt. The report considers an analysis using maximimum delays and does not filter paths based on slack threshold. It reports two paths per section and one expanded path per section.

```
report –type timing_violations \
  -analysis max –use_slack_threshold no \
  -limit_max_paths –yes \
  -max_paths 2 \
  -max_expanded_paths 1\
  timing_viol.txt
```

save

Tcl command; saves all changes made prior to this command. This includes changes made on constraints, options and sets.

```
save
```

Arguments

None

Example

The following script sets the maximum number of paths reported by list_paths to 10, reads an SDC file, and save both the option and the constraints into the design project:

```
set_options -limit_max_paths 10
read_sdc somefile.sdc
save
```

See Also

set_options

set_options

SmartTime-specific Tcl command; sets options for timing analysis. Some options will also affect timing-driven place-and-route. The same parameters can be changed in the SmartTime Options dialog box in the SmartTime GUI.

```
set_options
 [-max_opcond value ]
 [-min_opcond value]
 [-interclockdomain_analysis value]
 [-use_bibuf_loopbacks value]
 [-enable_recovery_removal_checks value]
 [-break_at_async value]
 [-filter_when_slack_below value]
 [-filter_when_slack_above value]
 [-remove_slack_filters]
 [-limit_max_paths value]
 [-expand_clock_network value]
 [-expand_parallel_paths value]
 [-analysis_scenario value]
 [-tdpr_scenario value]
 [-reset]
```

Arguments

-max_opcond `value`

Sets the operating condition to use for Maximum Delay Analysis. The following table shows the acceptable values for this argument. Default is *slow_lv*.

Value	Description
slow_lv	Use slow_lv conditions for Maximum Delay Analysis
slow_lv_lt	Use slow_lv_lt conditions for Maximum Delay Analysis
fast_hv_lt	Use fast_hv_lt conditions for Maximum Delay Analysis

-min_opcond `value`

Sets the operating condition to use for Minimum Delay Analysis. The following table shows the acceptable values for this argument. Default is *fast_hv_lt*.

Value	Description
fast_hv_lt	Use fast_hv_lt conditions for Maximum Delay Analysis
slow_lv_lt	Use slow_lv_lt conditions for Maximum Delay Analysis
slow_lv	Use slow_lv conditions for Maximum Delay Analysis

-interclockdomain_analysis `value`

Enables or disables inter-clock domain analysis. Default is *yes*.

Value	Description

Value	Description
yes	Enables inter-clock domain analysis
no	Disables inter-clock domain analysis

-use_bibuf_loopbacks *value*

Instructs the timing analysis whether to consider loopback path in bidirectional buffers (D->Y, E->Y)as false-path {no}. Default is *yes*; i.e., loopback are false paths.

Value	Description
yes	Enables loopback in bibufs
no	Disables loopback in bibufs

-enable_recovery_removal_checks *value*

Enables recovery checks to be included in max-delay analysis and removal checks in min-delay analysis. Default is *yes*.

Value	Description
yes	Enables recovery and removal checks
no	Disables recovery and removal checks

-break_at_async *value*

Specifies whether or not timing analysis is allowed to cross asynchronous pins (clear, reset of sequential elements). Default is *no*.

Value	Description
yes	Enables breaking paths at asynchronous ports
no	Disables breaking paths at asynchronous ports

-filter_when_slack_below *value*

Specifies a minimum slack value for paths reported by list_paths. Not set by default.

-filter_when_slack_above *value*

Specifies a maximum slack value for paths reported by list_paths. Not set by default.

-remove_slack_filters

Removes the slack minimum and maximum set using -filter_when_slack_below and filter_when_slack_above.

-limit_max_paths *value*

Specifies the maximum number of paths reported by list_paths. Default is *100*.

-expand_clock_network *value*

Specify whether or not clock network details are reported in expand_path. Default is *yes*.

Value	Description

Value	Description
yes	Enables expanded clock network information in paths
no	Disables expanded clock network information in paths

-expand_parallel_paths *value*

Specify the number of parallel paths {paths with the same ends} to include in expand_path. Default is *1*.

-analysis_scenario *value*

Specify the constraint scenario to be used for timing analysis. Default is *Primary*, the default scenario.

-tdpr_scenario *value*

Specify the constraint scenario to be used for timing-driven place-and-route. Default is Primary, the default scenario.

-reset

Reset all options to the default values, except those for analysis and TDPR scenarios, which remain unchanged.

Examples

The following script commands the timing engine to use best operating conditions for both max-delay analysis and min-delay analysis:

```
set_options -max_opcond {best} -min_opcond {best}
```

The following script changes the scenario used by timing-driven place-and-route and saves the change in the Libero project for place-and-route tools to see the change.

```
set_options -tdpr_scenario {My_TDPR_Scenario}
```

See Also

save

Programming and Configuration Tcl Commands

configure_design_initialization_data

This Tcl command sets the parameter values needed for generating initialization data.

```
configure_design_initialization_data
   -second_stage_start_address {<valid_snvm_address>} \
   -third_stage_start_address {<valid_address_for_third_stage_memory_type>} \
   -third_stage_memory_type {<UPROM | SNVM | SPIFLASH_NONAUTH >}\
   -third_stage_spi_clock_divider { 1 | 2 | 4 | 6} \
   -init_timeout {<int_between_1_and_128_seconds>}
```

Arguments

`-second_stage_start_address`

String parameter for the start address of the second stage initialization client.

Specified as a 32-bit hexadecimal string.

The first stage client is always placed in sNVM, so it must be a valid sNVM address aligned on a page boundary.

There are 221 sNVM pages and each page is 256 bytes long, so the address will be between 0 and DC00.

Notes:

Although the actual size of each page is 256 bytes, only 252 bytes are available to the user.

The first stage initialization client is always added to SNVM at 0xDC00 (page 220). So the valid addresses for the second stage initialization client are 0x0 (page 0) to 0xDB00 (page 219).

`-third_stage_start_address`

String parameter for the start address of the third stage initialization client.

Specified as a 32-bit hexadecimal string, and must be one of the following:

— valid sNVM address aligned on a page boundary

— valid UPROM address aligned on a block boundary

— valid SPIFLASH address

`-third_stage_memory_type`

The memory where the third stage initialization client will be placed.

The value can be UPROM, SNVM, or SPIFLASH_NONAUTH. The default is UPROM.

This parameter determines the valid value for parameter ‘third_stage_start_address’.

`-third_stage_spi_clock_divider`

The value can be 1, 2, 4, or 6. The default value is 1.

`-init_timeout`

Timeout value in seconds. Initialization is aborted if it does not complete before timeout expires.

The value can be between 1 and 128. The default value is 128.

Example

```
configure_design_initialization_data
   -second_stage_start_address 200 \
-third_stage_start_address 400 \
-third_stage_memory_type UPROM \
-third_stage_spi_clock_divider 4 \
-init_timeout 120
```

See Also

generate_design_initialization_data

configure_ram

This Tcl command uses the RAM configuration file to configure RAM.

```
configure_ram
  -cfg file <RAM configuration file>
```

Arguments

`-cfg_file` *`file`*

`file` is a valid configuration file to configure RAM.

Supported Families

PolarFire

set_client (for RAM)

```
set_client \
   -logical_instance_name {} \
   -content_type {MEMORY_FILE | INITIALIZE_WITH_ZERO} \
   -memory_file_format {} \
   -memory_file {} \
   -content_type_changed {1}
```

Arguments

`-logical_instance_name`

Hierarchical instance name of the logical RAM that the client will initialize.

`-content_type`

The -content_type can be MEMORY_FILE or INITIALIZE_WITH_ZERO.

MEMORY_FILE – Client will be initialized with content from the memory file.

INITIALIZE_WITH_ZERO – Client will be initialized with zeros.

`-memory_file_format`

For RAM blocks from PF_SRAM_AHBL_AXI core, the supported file format is “INTEL_HEX”. For all others, the supported file formats are “INTEL_HEX” and “MOTOROLA_S”.

`-memory_file`

Path of the imported memory file. This can be absolute, or relative to the project.

`-content_type_changed {1}`

If client information is modified by the user in Fabric RAMs, the value must be "1".

Example

The following example shows the set_client Tcl command for RAM.

```
set_client \
   -logical_instance_name {TOP/SD1_0/dplsramcomp1_0} \
   -content_type {MEMORY_FILE} \
   -memory_file_format {} \
   -memory_file
{D:/local_z_folder/work/libero_projects/g5/mint_gfad_spi_done/ram_client_1.hex} \
   -content_type_changed {1}
```

configure_snvm

Tcl command; configures sNVM from the specified configuration file.

```
configure_snvm -cfg_file  file
```

Arguments

`-cfg_file` *`file`*

`file` is a valid configuration file to configure sNVM.

See Also

Configure sNVM

Sample sNVM Configuration File

```
set_plain_text_client \
  -client_name {pt_A} \
  -number_of_bytes 64 \
  -content_type {MEMORY_FILE} \
  -content_file_format {Microsemi-Binary 8/16/32 bit} \
  -content_file {C:/local_z_folder/work/memory files/binary8x16.mem} \
  -start_page 0 \
  -use_for_simulation 0 \
  -reprogram 1 \
  -use_as_rom 0
set_plain_text_client \
  -client_name {pt_client} \
  -number_of_bytes 64 \
  -content_type {MEMORY_FILE} \
  -content_file_format {Microsemi-Binary 8/16/32 bit} \
  -content_file {C:/local_z_folder/work/memory files/binary32X16.mem} \
  -start_page 2 \
  -use_for_simulation 0 \
  -reprogram 1 \
  -use_as_rom 0
set_plain_text_client \
  -client_name {pt_client_16bit} \
  -number_of_bytes 32 \
  -content_type {MEMORY_FILE} \
  -content_file_format {Microsemi-Binary 8/16/32 bit} \
  -content_file {C:/local_z_folder/work/memory files/binary16X16.mem} \
  -start_page 1 \
  -use_for_simulation 0 \
  -reprogram 1 \
  -use_as_rom 0
```

```
set_plain_text_client \
  -client_name {INIT_STAGE_1_SNVM_CLIENT} \
  -number_of_bytes 504 \
  -content_type {MEMORY_FILE} \
  -content_file_format {Microsemi-Binary 8/16/32 bit} \
  -content_file {designer\top\top_init_stage_1_snvm.mem} \
  -start_page 219 \
  -use_for_simulation 0 \
  -reprogram 1 \
  -use_as_rom 0
set_plain_text_client \
  -client_name {pt_B} \
  -number_of_bytes 1 \
  -content_type {STATIC_FILL} \
  -content_file_format {Microsemi-Binary 8/16/32 bit} \
  -content_file {} \
  -start_page 3 \
  -use_for_simulation 0 \
  -reprogram 1 \
  -use_as_rom 0
```

See Also

set_plain_text_client

set_plain_text_auth_client

set_cipher_text_auth_client

set_usk_client

configure_spiflash

This Tcl command configures SPI Flash Memory from the specified SPI Flash Memory configuration file.

```
configure_spiflash -cfg_file file
```

Arguments

`-cfg_file` *`file`*

Specify a valid configuration file to configure SPI Fash.

`file` is the SPI Flash Memory configuration file. *`file`* can be an absolute path to the SPI Flash Memory configuration file or it can be a path relative to a Tcl file that includes the command. After running this command, the new configuration is saved as a project `spiflash.cfg` file.

See Also

Configure SPI Flash

Sample SPI Flash Configuration File

```
set_auto_update_mode {0}
set_manufacturer {Macronix}
set_client \
  -client_name {vzcx} \
  -client_type {FILE_SPI} \
  -content_type {MEMORY_FILE} \
  -content_file {..\..\..\..\memory files\spi_bitstream.spi} \
  -start_address {2561} \
  -client_size {388} \
  -program {1}
set_client \
  -client_name {golden} \
  -client_type {FILE_SPI_GOLDEN} \
  -content_type {MEMORY_FILE} \
  -content_file {C:\local_z_folder\work\memory files\spi_bitstream.spi} \
  -start_address {1042} \
  -client_size {389} \
  -program {1}
set_client \
  -client_name {INIT_STAGE_3_SPI_CLIENT} \
  -client_type {INIT} \
  -content_type {MEMORY_FILE} \
  -content_file {C:\local_z_folder\work\libero_projects\g5\SNVM_TEST_top_uic.bin} \
  -start_address {4096} \
  -client_size {4124} \
  -program {1}
```

configure_uprom

Tcl command; configures uPROM from the specified configuration file.

```
configure_uprom -cfg_file file
```

Arguments

`-cfg_file` *`file`*

`file` is a valid configuration file to configure uPROM.

See Also

Configure uPROM

Sample uPROM Configuration File

```
set_data_storage_client \
  -client_name {client1_from_elsewhere} \
  -number_of_words 37 \
  -use_for_simulation {0} \
  -content_type {MEMORY_FILE} \
  -memory_file_format {Microsemi-Binary} \
  -memory_file {C:/local_z_folder/work/memory files/sar_86586_uprom.mem} \
  -base_address 1500
set_data_storage_client \
  -client_name {large_1} \
  -number_of_words 100 \
  -use_for_simulation {0} \
  -content_type {STATIC_FILL} \
  -base_address 5000
```

export_spiflash_image

This Tcl command exports a SPI Flash image file to a specified directory.

```
export_spiflash_image -file_name {name of file} -export_dir {absolute path to folder location}
```

Arguments

`-file_name name of file`

The name of the image file.

`-export_dir absolute path to folder location`

Folder/directory location.

See Also

Export Flash Image

generate_design_initialization_data

This Tcl command creates the memory files on disk, adds the initialization clients to the target memories, and writes the configuration files to disk.

This command also runs validation on the saved configuration files and writes out errors (if any) in the log. This command causes the UI of the Configure Design Initialization Data and Memories tool to refresh and show the latest configuration and validation errors (if any) in the tables.

This command takes no parameters.

```
generate_design_initialization_data
```

See Also

configure_design_initialization_data

generate_initialization_mem_files

This Tcl command sets the parameter values needed for generating memory files to be used with design initialization clients.

```
generate_initialization_mem_files
   -second_stage_start_address {<valid_snvm_address>} \
   -third_stage_start_address {<valid_address_for_third_stage_memory_type>} \
   -third_stage_memory_type {<UPROM | SNVM | SPIFLASH_NONAUTH >}\
   -third_stage_spi_clock_divider{ 1 | 2 | 4 | 6} \
   -init_timeout { <int_between_1_and_128_seconds>}\
   -custom_cfg_file {<valid_user_specified_configuration_file>}
```

Arguments

`-second_stage_start_address`

String parameter for the start address of the second stage sNVM initialization client.

Specified as a 32-bit hexadecimal string.

The second stage client is always placed in sNVM, so it must be a valid sNVM address aligned on a page boundary.

This address will be between 0 and DB00. There are 221 sNVM pages and each page is 256 bytes long. The last two pages are reserved for the first stage initialization client so they are not available for the second stage initialization client.

`-third_stage_memory_type`

The memory where the third stage initialization client will be placed.

The value can be UPROM, SNVM, or SPIFLASH_NONAUTH. The default is sNVM.

This parameter determines the valid value for parameter ‘third_stage_start_address’.

`-third_stage_start_address`

String parameter for the start address of the third stage initialization client.

Specified as a 32-bit hexadecimal string, and must be one of the following:

- valid sNVM address aligned on a page boundary
- valid UPROM address aligned on a block boundary
- valid SPIFLASH address

`-third_stage_spi_clock_divider`

The value can be 1, 2, 4, or 6. The default value is 1.

`-init_timeout`

Timeout value in seconds. Initialization is aborted if it does not complete before timeout expires.

The value can be between 1 and 128. The default value is 128.

`-custom_cfg_file`

Specifies the user_specified configuration file to be loaded in.

Example

```
generate_initialization_mem_files \
   -second_stage_start_address 200 \
   -third_stage_memory_type UPROM \
   -third_stage_start_address 400 \
   -third_stage_spi_clock_divider 6 \
   -init_timeout 120 \
   -custom_cfg_file {D:\test\my.txt}
```

See Also

Design and Memory Initialization

GENERATEPROGRAMMINGFILE

GENERATEPROGRAMMINGFILE is a command tool used in the configure_tool and run_tool commands. The configure_tool -name {GENERATEPROGRAMMINGFILE} Tcl command configures tool options. The run_tool Tcl command runs the specified tool with the options specified in configure_tool.

```
configure_tool \
   -name {GENERATEPROGRAMMINGFILE} \
   -params {program_fabric:true|false} \
   -params {program_security:true|false} \
   -params {program_snvm:true|false}
run_tool -name {GENERATEPROGRAMMINGFILE}
```

The following tables list the parameter names and values.

configure_tool –name {GENERATEPROGRAMMINGFILE} parameter:value pair

Name	Value	Description

Name	Value	Description
program_fabric	true | false	Include fabric component in the programming bitstream.
program_security	true | false	Include custom security component in the programming bitstream ("true" only if custom security was defined).
program_snvm	true | false	Include sNVM component in the programming bitstream ("true" only if sNVM available in the design).

run_tool –name {GENERATEPROGRAMMINGFILE}

This command takes no parameters.

PROGRAM_SPI_FLASH_IMAGE

This Tcl command used in configure_tool and run_tool to program SPI Flash Image with configured parameters.

```
configure_tool \
   -name {PROGRAM_SPI_FLASH_IMAGE} \
   -params {spi_flash_prog_action: PROGRAM_SPI_FLASH}
run_tool \
   -name {PROGRAM_SPI_FLASH_IMAGE}
```

set_auto_update_mode

This command enables or disables auto update.

```
set_auto_update_mode {0|1}
```

If `set_auto_update_mode` is 0, auto update is disabled. If `set_auto_update_mode` is 1, auto update is enabled.

set_cipher_text_auth_client

This Tcl command is added to the sNVM .cfg file that is given as the parameter to the configure_snvm command.

Cipher-text Authenticated clients have 236 bytes available for user data in each page of sNVM.

```
set_cipher_text_auth_client
   -client_name {<name>}
   -number_of_bytes <number>
   -content_type {MEMORY_FILE | STATIC_FILL}
   -content_file_format {Microsemi-Binary 8/16/32 bit}
   -content_file {<path>}
   -start_page <number>
   -use_for_simulation 0
```

```
    -reprogram 0 | 1
    -use_as_rom 0 | 1
```

Arguments

`-client_name`

The name of the client. Needs to start with an alphabetic letter. Underscores and numerals are allowed at all positions other than the first.

`-number_of_bytes`

The size of the client specified in bytes.

`-content_type`

Source of data for the client. This can either be a memory file, or all zeros. Allowed values are MEMORY_FILE or STATIC_FILL

`-content_file_format`

Only ‘Microsemi-Binary 8/16/32 bit’ is supported at this time.

`-content_file`

Path of the memory file. This can be absolute, or relative to the project.

`-start_page`

The page number in sNVM where data for this client will be placed.

`-use_for_simulation`

Only value 0 is allowed.

`-reprogram`

Boolean field; specifies whether the client will be programmed into the final design or not. Possible values are 0 or 1.

`-use_as_rom 0`

Boolean field; specifies whether the client will allow only reads, or both read and writes. Possible values are 0 or 1.

Example

```
set_cipher_text_auth_client \
   -client_name {c} \
   -number_of_bytes 12 \
   -content_type {MEMORY_FILE} \
   -content_file_format {Microsemi-Binary 8/16/32 bit} \
   -content_file {D:/local_z_folder/work/memory_files/binary8x12.mem} \
   -start_page 3 \
   -use_for_simulation 0 \
   -reprogram 1 \
```

See Also

set_plain_text_client

set_plain_text_auth_client

set_usk_client

set_client

This command specifies the client that will be added to SPI Flash Memory. . This command is added to the SPI Flash Memory configuration file that is given as the parameter to the configure_spiflash command.

```
set_client \
    -client_name {} \
    -client_type {FILE_SPI | FILE_SPI_GOLDEN | FILE_SPI_UPDATE |FILE_DATA_STORAGE_INTELHEX
\
    -content_type {MEMORY_FILE | STATIC_FILL} \
    -content_file {} \
    -start_address {} \
    -client_size {} \
    -program {0|1}
```

Arguments

`-client_name`

The name of the client. Maximum of 32 characters, letters or numbers or "-" or "_".

`-client_type`

The `-client_type` can be `FILE_SPI`, `FILE_SPI_GOLDEN`, `FILE_SPI_UPDATE` or `FILE_DATA_STORAGE_INTELHEX`.

`FILE_SPI` – SPI Bitstream

`FILE_SPI_GOLDEN` – Recovery/Golden SPI Bitstream

`FILE_SPI_UPDATE` – Auto Update SPI Bitstream; available only if Auto Update is enabled. See set_auto_update_mode.

`FILE_DATA_STORAGE_INTELHEX` - Data Storage client

`-content_type`

The `-content_type` can be MEMORY_FILE or STATIC_FILL.

`MEMORY_FILE` – content memory file must be specified

`STATIC_FILL` – client memory will be filled with 1s, no content memory file

`-content_file`

Absolute or relative path to the content memory file.

`-start_address`

The client start address. Note that some space is reserved for the SPI Flash Memory directory.

`-client_size`

Client's size in bytes. If a content file is specified, the size must be equal to or larger than the file size.

`-program {1}`

Note: Only `program | 1` is supported in this release.

Examples

The following examples shows the `set_client` Tcl command for SPI Flash.

Absolute path

```
set_client \
   -client_name {golden} \
   -client_type {FILE_SPI_GOLDEN} \
   -content_type {MEMORY_FILE} \
   -content_file {E:\top_design_ver_1.spi} \
   -start_address {1024} \
   -client_size {9508587} \
   -program {1}

set_client \
   -client_name {ds} \
   -client_type {FILE_DATA_STORAGE_INTELHEX} \
   -content_type {MEMORY_FILE} \
   -content_file {E:\intel_hex.hex} \
   -start_address {9509611} \
   -client_size {128} \
   -program {1}
```

Relative path

```
set_client \
   -client_name {golden} \
   -client_type {FILE_SPI_GOLDEN} \
   -content_type {MEMORY_FILE} \
   -content_file {.\..\..\top_design_ver_1.spi} \
   -start_address {1024} \
   -client_size {9508587} \
   -program {1}

set_client \
   -client_name {ds} \
   -client_type {FILE_DATA_STORAGE_INTELHEX} \
   -content_type {MEMORY_FILE} \
   -content_file {.\..\..\intel_hex.hex} \
   -start_address {9509611} \
   -client_size {128} \
   -program {1}
```

set_data_storage_client

This Tcl command is added to the .cfg file, which will then be given as the parameter to the configure_uprom command.

```
set_data_storage_client \
   -client_name {<name> } \
   -number_of_words <number> \
   -content_type {MEMORY_FILE | STATIC_FILL} \
   -memory_file_format {Microsemi-Binary} \
   -memory_file {<path>} \
   -base_address {<hexadecimal_string>} \
   -use_for_simulation {0} \
```

Arguments

`-client_name`

The name of the client. Must start with an alphabetic letter. Underscores and numerals are allowed at all positions other than the first.

`-number_of_bytes`

The size of the client specified in number of words.

`-content_type`

Source of data for the client. This can either be a memory file, or all zeros. Allowed values are `MEMORY_FILE` or `STATIC_FILL`.

`MEMORY_FILE` – content memory file must be specified

`STATIC_FILL` – client memory will be filled with 1s, no content memory file

`-memory_file_format`

Only ‘Microsemi-Binary’ is supported at this time.

`-content_file`

Path of the memory file. This can be absolute, or relative to the project.

`-base_address`

Hexadecimal address where the first byte of user data will be placed.

`-use_for_simulation`

Only value 0 is allowed.

Example

```
set_data_storage_client \
   -client_name {client1_from_elsewhere_new_MMMMMMM} \
   -number_of_words 57 \
   -use_for_simulation {0} \
   -content_type {MEMORY_FILE} \
   -memory_file_format {Microsemi-Binary} \
   -memory_file {D:/local_z_folder/work/memory_files/sar_86586_uprom.mem} \
   -base_address 0
```

set_manufacturer

This command specifies the manufacturer for the SPI Flash device.

```
set_manufacturer {MICRON | SPANSION | Macronix | Winbond }
```

The value for the `set_manufacturer` command must be one of the following:

- MICRON
- SPANSION
- Macronix
- Winbond

See the following table for details about the supported SPI Flash devices.

Mfg Part Number	Memory Capacity	Manufacturer	Sector Size
MT25QL01GBBB8ESF-0SIT	1 GB	MICRON	4 KB
S25FL512SAGMFI011	512 MB	SPANSION	256 KB
MX66L51235FMI-10G	512 MB	Macronix	4 KB
W25Q256FVFIG	256 MB	Winbond	4 KB
Note: Microsemi currently supports only the devices listed above.			

See Also

Microsemi Factory Access Policy

set_plain_text_auth_client

This Tcl command is added to the sNVM .cfg file that is given as the parameter to the configure_snvm command.

Plain-text Authenticated clients have 236 bytes available for user data in each page of sNVM.

```
set_plain_text_auth_client
   -client_name {<name>}
   -number_of_bytes <number>
   -content_type {MEMORY_FILE | STATIC_FILL}
   -content_file_format {Microsemi-Binary 8/16/32 bit}
   -content_file {<path>}
   -start_page <number>
   -use_for_simulation 0
   -reprogram 0 | 1
   -use_as_rom 0 | 1
```

Arguments

`-client_name`

The name of the client. Needs to start with an alphabetic letter. Underscores and numerals are allowed at all positions other than the first.

`-number_of_bytes`

The size of the client specified in bytes.

`-content_type`

Source of data for the client. This can either be a memory file, or all zeros. Allowed values are MEMORY_FILE or STATIC_FILL

`-content_file_format`

Only ‘Microsemi-Binary 8/16/32 bit’ is supported at this time.

`-content_file`

Path of the memory file. This can be absolute, or relative to the project.

`-start_page`

The page number in sNVM where data for this client will be placed.

`-use_for_simulation`

Only value 0 is allowed.

`-reprogram`

Boolean field; specifies whether the client will be programmed into the final design or not. Possible values are 0 or 1.

`-use_as_rom 0`

Boolean field; specifies whether the client will allow only reads, or both read and writes. Possible values are 0 or 1.

Example

```
set_plain_text_auth_client \
   -client_name {b} \
   -number_of_bytes 12 \
   -content_type {MEMORY_FILE} \
   -content_file_format {Microsemi-Binary 8/16/32 bit} \
   -content_file {D:/local_z_folder/work/memory_files/binary8x12.mem} \
   -start_page 2 \
   -use_for_simulation 0 \
   -reprogram 1 \
   -use_as_rom 0
```

See Also

set_plain_text_client

set_cipher_text_auth_client

set_usk_client

set_plain_text_client

This Tcl command is added to the sNVM .cfg file that is given as the parameter to the configure_snvm command.

Plain-text Non-Authenticated clients have 252 bytes available for user data in each page of sNVM.

```
set_plain_text_client
   -client_name {<name>}
   -number_of_bytes <number>
   -content_type {MEMORY_FILE | STATIC_FILL}
   -content_file_format {Microsemi-Binary 8/16/32 bit}
   -content_file {<path>}
```

```
    -start_page <number>
    -use_for_simulation 0
    -reprogram 0 | 1
    -use_as_rom 0 | 1
```

Arguments

`-client_name`

The name of the client. Needs to start with an alphabetic letter. Underscores and numerals are allowed at all positions other than the first.

`-number_of_bytes`

The size of the client specified in bytes.

`-content_type`

Source of data for the client. This can either be a memory file, or all zeros. Allowed values are MEMORY_FILE or STATIC_FILL

`-content_file_format`

Only ‘Microsemi-Binary 8/16/32 bit’ is supported at this time.

`-content_file`

Path of the memory file. This can be absolute, or relative to the project.

`-start_page`

The page number in sNVM where data for this client will be placed.

`-use_for_simulation`

Only value 0 is allowed.

`-reprogram`

Boolean field; specifies whether the client will be programmed into the final design or not. Possible values are 0 or 1.

`-use_as_rom 0`

Boolean field; specifies whether the client will allow only reads, or both read and writes. Possible values are 0 or 1.

Example

```
set_plain_text_client \
   -client_name {a} \
   -number_of_bytes 12 \
   -content_type {MEMORY_FILE} \
   -content_file_format {Microsemi-Binary 8/16/32 bit} \
   -content_file {D:/local_z_folder/work/memory_files/binary8x12.mem} \
   -start_page 1 \
   -use_for_simulation 0 \
   -reprogram 1 \
   -use_as_rom 0
```

See Also

set_plain_text_auth_client

set_cipher_text_auth_client

set_usk_client

set_usk_client

This Tcl command is added to the sNVM .cfg file that is given as the parameter to the configure_snvm command.

The USK client is required if sNVM has one or more clients of type ‘Authenticated’.

```
set_cipher_text_auth_client
   -start_page <number>
   -key <Hexadecimal string of size 24>
   -use_for_simulation 0 | 1
   -reprogram 0 | 1
```

Arguments

`-start_page`

The page number in sNVM where data for this client will be placed.

`-key`

A string of 24 hexadecimal characters.

`-use_for_simulation`

Boolean field specifies whether the client will be used for simulation or not. Possible values are 0 or 1.

`-reprogram`

Boolean field; specifies whether the client will be programmed into the final design or not. Possible values are 0 or 1.

Example

```
set_usk_client \
   -start_page 4 \
   -key {D8C8831F3A2F72EDC569503F} \
   -use_for_simulation 0 \
   -reprogram 1
```

See Also

set_plain_text_client

set_plain_text_auth_client

set_cipher_text_auth_client

COMPILE

Note: COMPILE is a valid tool name when EDIF design source files are used in the Enhanced Constraint Flow. For non-EDIF or HDL designs using the Enhanced Constraint Flow, the COMPILE step is subsumed under the SYNTHESIZE tool.

COMPILE is a command tool used in configure_tool and run_tool. Configure_tool allows you to configure the tool's parameters and values prior to executing the tool. Run_tool executes the tool with the configured parameters.

To compile the design in Libero SoC, first configure the compile tool with the configure_tool command, and then execute the COMPILE command with the run_tool command.

```
configure_tool -name {COMPILE}
-params {name:value}
[-params {name:value}]
run_tool -name {COMPILE}
```

The following tables list the parameter names and values.

configure_tool –name {COMPILE} parameter:value pair

Name	Value	Description
PDC_IMPORT_HARD ERROR	Boolean {true | false | 1 | 0}	Set to true or 1 if you want the COMPILE command to abort when errors are found in the

Name	Value	Description
		physical design constraints. Default is false.
BLOCK_PLACEMENT_CONFLICTS	String {ERROR|KEEP|LOCK|DISCARD}	Instructs the COMPILE engine what to do when the software encouters a placement conflict. When set to: ERROR - Compile errors out if any instance from a Designer block becomes unplaced. This is the default. KEEP - If some instances get unplaced for any reason, the non-conflicting elements remaining are preserved but not locked. Therefore, the placer can move them into another location if necessary. LOCK - If some instances get unplaced for any reason, the non-conflicting elements remaining are preserved and locked. DISCARD – Discards any placement from the block, even if there are no conflicts.
BLOCK_ROUTING_CONFLICTS	String {ERROR|KEEP|LOCK|DISCARD}	Instructs the COMPILE engine what to do when the software encounters a routing conflict. When set to: ERROR - Compile errors out if any route in any preserved net from a Designer block is deleted. This is the default. KEEP – If a route is removed from a net for any reason, the routing for the non-conflicting nets is kept unlocked. The router can re-route these nets. LOCK – If routing is removed from a net for any reason, the routing for the non-conflicting nets is kept as locked, and the router will not change them. DISCARD - Discards any routing from the block, even if there are no conflicts.
PA4_GB_MAX_RCLKINT_INSERTION	Integer	Specifies the maximum number of global nets that could be demoted to row-globals. Default is 16, Min is 0 and Max is 50.
PA4_GB_MIN_GB_FANOUT_TO_USE_RCLKINT	Integer	Specifies the Minimum fanout of global nets that could be demoted to row-globals. Default is 300. Min is 25 and Max is 5000.
PA4_GB_MAX_FANOUT_DATA_MOVE	Integer	Specifies the Minimum fanout of non-clock nets to be kept on globals. Default is 5000. Min is 300 and Max is 200,000.
PA4_GB_COUNT	Integer	The number of available global nets is reported. Minimum for all dies is “0”. Default and Maximum values are die-dependent: 005/010 die: Default = Max = 8 025/050/060/090/150 die: Default=Max=16 Note:
BLOCK_MODE	Boolean {true | false | 1 | 0}	Set to true or 1 when you have blocks in your design and you want to enable the Block mode. Set it to false or 0 if you don’t have blocks in your design. Default is false or 0.

run_tool –name {COMPILE} Parameter:value pair

Name	Value	Description
NONE		

## Example

```
configure_tool -name {COMPILE}
  -params {BLOCK_MODE:false}
  -params {BLOCK_PLACEMENT_CONFLICTS:ERROR}
  -params {BLOCK_ROUTING_CONFLICTS:ERROR}
  -params {PA4_GB_MAX_RCLKINT_INSERTION:16}\
  -params {PA4_GB_MIN_GB_FANOUT_TO_USE_RCLKINT:30}\
  -params {PA4_GB_MAX_FANOUT_DATA_MOVE:2000}\
  -params {PA4_GB_COUNT:8}\
  -params {PDC_IMPORT_HARDERROR:true}
run_tool –name {COMPILE} #Takes no parameters
```

Return

`configure_tool -name {COMPILE}`

Returns 0 on success and 1 on failure.

`run_tool -name {COMPILE}`

Returns 0 on success and 1 on failure.

CONFIGURE_CHAIN

CONFIGURE_CHAIN is a command tool used in run_tool. The command run_tool -name {CONFIGURE_CHAIN} takes a script file that contains FlashPro-specific Tcl commands and passes them to FlashPro Express for execution.

```
run_tool -name {CONFIGURE_CHAIN} –script {fpro_cmds.tcl}
```

fpro_cmds.tcl is a Tcl script that contains FlashPro-specific Tcl commands to configure JTAG chain. For details on JTAG chain programming Tcl commands, refer to the Tcl commands section in FlashPro's Online Help.

Do not include any FlashPro project-management commands such as open_project, save_project, or close_project in this *fpro_cmds.tcl* script file. The run_tool –name {CONFIGURE_CHAIN} command generates these project-management commands for you.

Note: For a new Libero project without a JTAG chain, executing this command causes Libero to first add the existing design device to the JTAG chain and then execute the commands from the FlashPro script. If, for example, the FlashPro script *fpro_cmds.tcl* contains commands to add four devices, executing the command run_tool –name {CONFIGURE_CHAIN} -script {*fpro_cmds.tcl*} will create a JTAG chain of the Libero design device and the four devices. For existing Libero projects that already have a JTAG chain, the command is executed on the existing JTAG chain.

## Example

`run_tool –name {CONFIGURE_CHAIN} –script {d:/fpro_cmds.tcl}`

#Example fpro_cmds.tcl command file for the –script parameter

```
add_actel_device \
  -file {./sd_prj/sp_g3/designer/impl1/sd1.stp} \
  -name {dev1}
```

```
enable_device -name {MPF300TS_ES} -enable 0
add_non_actel_device \
  -ir 2 \
  -tck 1.00 \
  -name {Non-Microsemi Device}
add_non_actel_device \
  -ir 2 \
  -tck 1.00 \
  -name {Non-Microsemi Device (2)}
remove_device -name {Non-Microsemi Device}
set_device_to_highz -name {MPF300TS_ES} -highz 1
add_actel_device \
  -device {MPF300TS_ES} \
  -name {MPF300TS_ES(3)}
select_libero_design_device -name {MPF300TS_ES(3)}
```

Return

Returns 0 on success and 1 on failure.

CONFIGURE_PROG_OPTIONS

CONFIGURE_PROG_OPTIONS is a command tool used in configure_tool. Configure_tool -name {CONFIGURE_PROG_OPTIONS} sets the programming options.

```
configure_tool -name {CONFIGURE_PROG_OPTIONS}
-params {design_version:<value>}
-params {silicon_signature:<value>}
```

The following table lists the parameter names and values.

configure_tool –name {CONFIGURE_PROG_OPTIONS} parameter:value pair

Name	Value	Description
design_version	Integer {0 through 65535}	Sets the design version. It must be greater than the Back level version in SPM Update Policy.
silicon_signature	Hex {<max length 8 Hex characters>}	32-bit (8 hex characters) silicon signature to be programmed into the device. This field can be read from the device using the JTAG USERCODE instruction.

Example

```
configure_tool -name {CONFIGURE_PROG_OPTIONS}\
  -params {design_version:255}

  -params {silicon_signature:abcdef}
```

Return

Returns 0 on success and 1 on failure.

PLACEROUTE

PLACEROUTE is a command tool used in configure_tool and run_tool. Configure_tool allows you to configure the tool's parameters and values prior to executing the tool. Run_tool executes the PLACEROUTE command tool with the configured parameters.

To place and route the design in Libero SoC, you must first configure the PLACEROUTE tool with the configure_tool command and then execute the PLACEROUTE command with the run_tool command.

```
configure_tool -name {PLACEROUTE}
-params {name:value}
-params {name:value}
-params {name:value}
-params {name:value}
run_tool -name {PLACEROUTE}
```

The following tables list the parameter names and values.

configure_tool –name {PLACEROUTE} parameter:value pair

Name	Value	Description
TDPR	Boolean {true | false | 1 | 0}	Set to true or 1 to enable Timing-Driven Place and Route. Default is 1.
PDPR	Boolean {true | false | 1 | 0}	Set to true or 1 to enable Power-Driven Place and Route. Default is false or 0.
EFFORT_LEVEL	Boolean {true | false | 1 | 0}	Set to true or 1 to enable High Effort Layout to optimize design performance. Default is false or 0.
INCRPLACEANDROUTE	Boolean {true | false | 1 | 0}	Set to true or 1 to use previous placement data as the initial placement for the next run. Default is false or 0.
REPAIR_MIN_DELAY	Boolean {true | false | 1 | 0}	Set to 1 to enable Repair Minimum Delay violations for the router when TDPR option is set to true or 1. Default is false.
NUM_MULTI_PASSES	Integer value {"1" through "25"}	Specifies the number of passes to run. The default is 5. Maximum is 25.
START_SEED_INDEX	Integer from "1" to "101"	Indicates the specific index into the array of random seeds which is to be the starting point for the passes. Its value should range from 1 to 100. If not specified, the default behavior is to continue from the last seed index which was used.
MULTI_PASS_LAYOUT	Boolean {true | false | 1 | 0}	Set to true or 1 to enable Multi-Pass Layout Mode for Place and Route. Default is false or 0.
MULTI_PASS_CRITERIA	{SLOWEST_CLOCK" | SPECIFIC_CLOCK | VIOLATIONS | TOTAL_POWER}	Specifies the criteria used to run multi-pass layout: • SLOWEST ClOCK: Use the slowest clock frequency in the design in a given pass as the performance reference for the layout pass.

Name	Value	Description
		• SPECIFIC_CLOCK: Use a specific clock frequency as the performance reference for all layout passes. • VIOLATIONS: Use the pass that best meets the slack or timing-violations constraints. This is the default. • TOTAL POWER:Specifies the best pass to be the one that has the lowest total power (static + dynamic) out of all layout passes.
SPECIFIC_CLOCK	{Name_of_clock}	Applies only when MULTI_PASS_CRITERIA is set to SPECIFIC_CLOCK. It specifies the name of the clock in the design used for Timing Violation Measurement.
DELAY_ANALYSIS	max | min	Used only when MULTI_PASS_CRITERIA is set to "VIOLATIONS". Specifies the type of timing violations (slacks) to be examined. The default is 'max'. • max: Use timing violations (slacks) obtained from maximum delay analysis • min: Use timing violations (slacks) obtained from minimum delay analysis.
STOP_ON_FIRST_PASS	Boolean {true | false | 1 | 0}	Applies only when MULTI_PASS_CRITERIA is set to "VIOLATIONS". It stops performing remaining passes if all timing constraints have been met (when there are no negative slacks reported in the timing violations report). Note: The type of timing violations (slacks) used is determined by the 'DELAY_ANALYSIS ' parameter.
SLACK_CRITERIA	{WORST_SLACK | TOTAL_NEGATIVE_SLACK}	Applies only when MULTI_PASS_CRITERIA is set to VIOLATIONS. Specifies how to evaluate the timing violations (slacks). The default is WORST_SLACK. • WORST_SLACK: The largest amount of negative slack (or least amount of positive slack if all constraints are met) for each pass is identified and then the largest value out of all passes will determine the best pass. This is the default. • TOTAL_NEGATIVE_SLACK: The sum of negative slacks from the first 100 paths for each pass in the Timing Violation report is identified. The largest value out of all passes will determine the best pass. If no negative slacks exist for a pass, then use the worst slack to evaluate that pass. Note: The type of timing violations (slacks) used is determined by the 'DELAY_ANALYSIS' parameter.

Name	Value	Description
RGB_COUNT	Integer from “1” to “18” (default is 18)	Allows an entity to override the placer's RGB/RCLK bandwidth constraint. This option is useful for Block Creation.
REPLICATION	Boolean {true | false | 1 | 0}	Set to true or 1 to enable Driver Replication. Default is 1.

run_tool –name {PLACEROUTE}

Example

```
configure_tool -name {PLACEROUTE}\
  -params {TDPR:true}\
  -params {PDPR:false}\
  -params {EFFORT_LEVEL:true}\
  -params {INCRPLACEANDROUTE:false}\
run_tool –name {PLACEROUTE} #Takes no parameters
```

Return

```
configure_tool -name {PLACEROUTE}
```

Returns 0 on success and 1 on failure.

```
run_tool -name {PLACEROUTE}
```

Returns 0 on success and 1 on failure.

SYNTHESIZE

SYNTHESIZE is a command tool used in configure_tool and run_tool. Configure_tool is a general-purpose Tcl command that allows you to configure a tool's parameters and values prior to executing the tool. The run_tool Tcl command then executes the specified tool with the configured parameters.

To synthesize your design in Libero SoC, you first configure the synthesize tool with the configure_tool command and then execute the command with the run_tool command.

```
configure_tool -name {SYNTHESIZE}
    -params {name:value}
    [-params {name:value}]
run_tool -name {SYNTHESIZE}
```

The following tables list the parameter names and values.

configure_tool –name {SYNTHESIZE} parameter:value pair

Name	Value	Description
CLOCK_ASYNC	Integer	Specifies the threshold value for asynchronous pin promotion to a global net. The default is 12.
CLOCK_GLOBAL	Integer	Specifies the threshold value for Clock pin promotion. The default is 2.
CLOCK_DATA	Integer value between 1000 and 200,000.	Specifies the threshold value for data pin promotion. The default is 5000.

Name	Value	Description
RAM_OPTIMIZED_FOR_POWER	Boolean {true | false | 1 | 0}	Set to true or 1 to optimize RAM for Low Power; RAMS are inferred and configured to ensure the lowest power consumption. Set to false or 0 to optimize RAM for High Speed at the expense of more FPGA resources.
RETIMING	Boolean {true | false | 1 | 0}	Set to true or 1 to enable Retiming during synthesis. Set to false or 0 to disable Retiming during synthesis.
SYNPLIFY_OPTIONS	String	Specifies additional synthesis-specific options. Options specified by this parameter override the same options specified in the user Tcl file if there is a conflict.
SYNPLIFY_TCL_FILE	String	Specifies the absolute or relative path name to the user Tcl file containing synthesis-specific options.
BLOCK_MODE	Boolean {true | false | 1 | 0}	Set to true or 1 when you have blocks in your design and you want to enable the Block mode. Set it to false or 0 if you don't have blocks in your design. Default is false or 0.
BLOCK_PLACEMENT_CONFLICTS	String {ERROR|KEEP|LOCK|DISCARD}	Instructs the COMPILE engine what to do when the software encounters a placement conflict. When set to: ERROR - Compile errors out if any instance from a Designer block becomes unplaced. This is the default. KEEP - If some instances get unplaced for any reason, the non-conflicting elements remaining are preserved but not locked. Therefore, the placer can move them into another location if necessary. LOCK - If some instances get unplaced for any reason, the non-conflicting elements remaining are preserved and locked. DISCARD – Discards any placement from the block, even if there are no conflicts.
BLOCK_ROUTING_CONFLICTS	String {ERROR|KEEP|LOCK|DISCARD}	Instructs the COMPILE engine what to do when the software encounters a routing conflict. When set to: ERROR - Compile errors out if any route in any preserved net from a Designer block is deleted. This is the default. KEEP – If a route is removed from a net for any reason, the routing for the non-conflicting nets is kept unlocked. The router can re-route these nets. LOCK – If routing is removed from a net for any reason, the routing for the non-conflicting nets is kept as locked, and the router will not change them. DISCARD - Discards any routing from the block, even if there are no conflicts.

Name	Value	Description
PA4_GB_COUNT	Integer	The number of available global nets is reported. Minimum for all dies is “0”. Default and Maximum values are die-dependent: 005/010 die: Default = Max = 8 025/050/060/090/150 die: Default=Max=16 RT4G075/RT4G150: Default=24, Max=48.
PA4_GB_MAX_RCLKINT_INSERTION	Integer	Specifies the maximum number of global nets that could be demoted to row-globals. Default is 16, Min is 0 and Max is 50.
PA4_GB_MIN_GB_FANOUT_TO_USE_RCLKINT	Integer	Specifies the Minimum fanout of global nets that could be demoted to row-globals. Default is 300. Min is 25 and Max is 5000.
SEQSHIFT_TO_URAM	Boolean {0,1}	Specifies whether the Sequential-Shift Registers are to be mapped to Registers or 64x12 RAMs. If set to 1 (the default), the logic mapping is to RAMs. If set to 0, the logic mapping is to Registers.
LANGUAGE_SYSTEM_VLOG	Boolean {true | false}	Set to true if the Verilog files contain System Verilog constructs.
LANGUAGE_VERILOG_2001	Boolean {true | false}	Set to true if Verilog files contain Verilog 2001 constructs.
LANGUAGE_VHDL_2008	Boolean {true | false}	Set to true if VHDL standard is VHDL 2008.

run_tool –name {SYNTHESIZE}

Example

```
configure_tool -name {SYNTHESIZE} -params {BLOCK_MODE:false}\
     -params {BLOCK_PLACEMENT_CONFLICTS:ERROR} –params\
     {BLOCK_ROUTING_CONFLICTS:ERROR} -params {CLOCK_ASYNC:12}\
     -params {CLOCK_DATA:5010} -params {CLOCK_GLOBAL:2} –params\
     –params {PA4_GB_MAX_RCLKINT_INSERTION:16} –params\
     {PA4_GB_MIN_GB_FANOUT_TO_USE_RCLKINT:299} –params\
     {RAM_OPTIMIZED_FOR_POWER:false} -params {RETIMING:false}
     -params {SYNPLIFY_OPTIONS:
     set_option -run_prop_extract 1;
     set_option -maxfan 10000;
     set_option -clock_globalthreshold 2;
     set_option -async_globalthreshold 12;
     set_option -globalthreshold 5000;
     set_option -low_power_ram_decomp 0;}\
     -params {SYNPLIFY_TCL_FILE:C:/Users/user1/Desktop/tclflow/synthesis/test.tcl}

run_tool –name {SYNTHESIZE} #Takes no parameters
```

Return

```
configure_tool -name {SYNTHESIZE}
```

Returns 0 on success and 1 on failure.

```
run_tool -name {SYNTHESIZE}
```

Returns 0 on success and 1 on failure.

VERIFYTIMING

VERIFYTIMING is a command tool used in run_tool. Run_tool passes a script file that contains timing-specific Tcl commands to the VERIFYTIMING command and executes it.

```
run_tool -name {VERIFYTIMING} –script {timing.tcl}
```

where

<timing.tcl> is a script that contains SmartTime-specific Tcl commands. You can include SmartTime-specific Tcl commands to create user path sets and to generate timing reports. See sample the Sample SmartTime Tcl Script below for details.

Example

```
run_tool –name {VERIFYTIMING} –script {<timing.tcl>}
```

Return

Returns 0 on success and 1 on failure.

Sample SmartTime Tcl Script <*timing.tcl*>

```
# Create user path set -from B_reg
create_set -name from_B_reg \
  -source {B_reg*[*]:CLK} \
  -sink {*}
# Create user set -from A, B, C
create_set -name from_in_ports \
  -source {A B C} \
  -sink {*}
# Generate Timing Reports
Report \
  -type timing \
  -analysis min \
  -format text \
  -max_paths 10 \
  -print_paths yes \
  -max_expanded_paths 10 \
  -include_user_sets yes \
  min_timing.rpt
# Export SDC
write_sdc -scenario {Primary} exported.sdc
#save the changes
save
```

SIMULATE

Use the run_tool command to run simulation with your default simulation tool.

```
#Run Pre-synthesis simulation
run_tool -name {SIM_PRESYNTH}
```

Return Value

Returns 0 on success and 1 on failure.

<timing.tcl> is a script that contains SmartTime-specific Tcl commands. You can include SmartTime-specific Tcl commands to create user path sets and to generate timing reports. See sample the Sample SmartTime Tcl Script below for details.

Example

```
run_tool –name {VERIFYTIMING} –script {<timing.tcl>}
```

Return

Returns 0 on success and 1 on failure.

Sample SmartTime Tcl Script <*timing.tcl*>

```
# Create user path set -from B_reg
create_set -name from_B_reg \
  -source {B_reg*[*]:CLK} \
  -sink {*}
# Create user set -from A, B, C
create_set -name from_in_ports \
  -source {A B C} \
  -sink {*}
# Generate Timing Reports
Report \
  -type timing \
  -analysis min \
  -format text \
  -max_paths 10 \
  -print_paths yes \
  -max_expanded_paths 10 \
  -include_user_sets yes \
  min_timing.rpt
# Export SDC
write_sdc -scenario {Primary} exported.sdc
#save the changes
save
```