PolarFire FPGA
Tcl Commands Reference Guide
Libero SoC v12.2

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

a **MICROCHIP** company

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

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.**

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

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:

- Exporting Tcl scripts
- extended_run_lib
- Tcl Commands as specified in this document

Tcl Commands and Supported Families

When we specify a family name, we refer to the device family and all its derivatives, unless otherwise specified.

See Supported Families in the Tcl command help topics for the families supported for a specific Tcl command.

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:
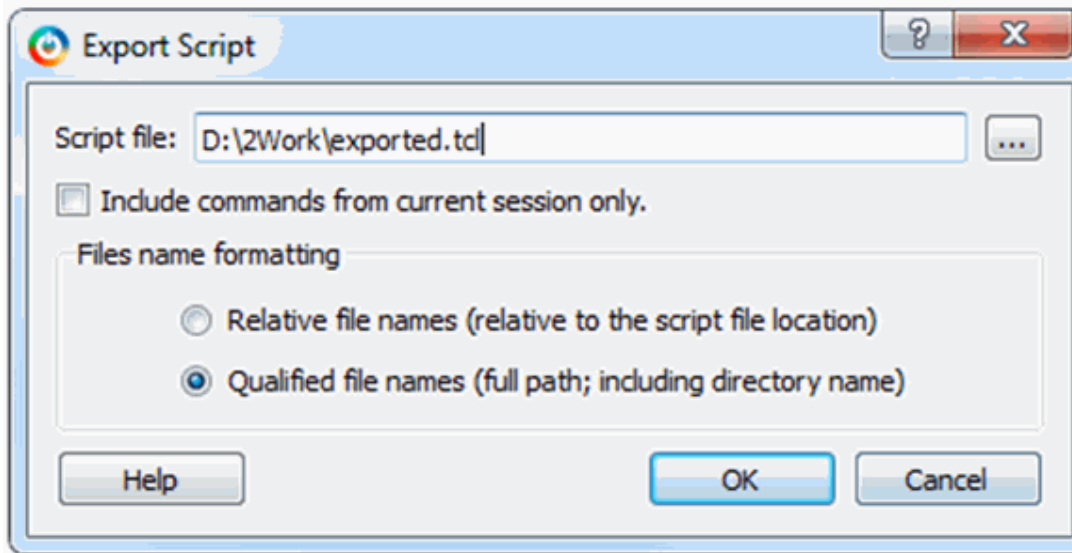
Figure 1 · Script Export Options

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.
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.

Exceptions

None

See Also

Place and Route - PolarFire
Multiple Pass Layout - PolarFire

See the online help for more information.

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}
```

How to Derive Required Part Information from A "Part Number"

To use Tcl Commands such as set_device or new_design; certain part information items must be specified. Many of these items can be derived from the "Part Number" you have chosen. For example, suppose the Part Number is: **MPF300XT-1FCG784I**

- -family <family name>
  The <family name> usually known, e.g.
  ```
  -family {PolarFire}
  ```

- -die <die name>
  From the Part Number, the characters before the "-": **MPF300XT**-1FCG784I
  ```
  -die {MPF300XT}
  ```

- -speed <speed grade>
  If there is a digit immediately after the "-", -<digit> will be the <speed grade> value (preceeded by a "-"). In this case: MPF300XT-**1**FCG784
  ```
  -speed {-1}
  ```
    - NOTE: If there is no digit, the default speed grade is STD.

- -package <package name>
  The next sequence of letters, followed by a sequence of digits will constitute the package type and "size".
  NOTE: If there is a trailing letter after the <digits>; this letter is **not** part of the <package name>; but is rather part of the <part range> (see below).
    - For PolarFire, this combination will simply constitute the <package name> e.g.: MPF300XT-1**FCG784**I
      ```
      -package {FCG784}
      ```

- -part_range <part range>
  The last letter (if any) will indicate the <part_range> according to the following table:

last letter	expansion value
I	IND
E	EXT
M	MIL
<none>	COM

- In this case: MPF300XT-1FCG784**I**
  ```
  -part_range {IND}
  ```

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. See the online help for more information.

```
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
```

change_vault_location

Tcl command; changes the location of the vault.

Note: This command overrides the vault location for all projects.

```
change_vault_location \
-location location
```

Arguments

```
-location location
```

Specifies the new vault location. Value must be a file path.It is mandatory

Examples

```
change_vault_location -location {../vault}
```

See Also

[Tcl Command Documentation Conventions](#)

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 com mand; 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

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

close_design
new_design
open_design

See the online help for more information.

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_core

Tcl command; modifies the configuration of an existing core component in the SmartDesign. This command works for core components created for different types of cores namely, Sg cores, System Builder cores and Direct cores.

Limitations: The command does not work for SmartFusion2/IGLOO2 System Builder components, SmartFusion2 MSS component, and RTG4 PCIE_SERDES_IF_INIT(RTG4 High Speed Serial Interface 1 - EPCS and XAUI - with Initialization), NPSS_SERDES_IF_INIT(RTG4 High Speed Serial Interface 2 - EPCS and XAUI - with Initialization) and RTG4FDDRC_INIT(RTG4 DDR Memory Controller with initialization) core components.

```
configure_core \
-component_name component_name \
-params core_parameters
```

Arguments

-component_name *component_name*

Specifies the name of the component to be configured. It is mandatory.

-params *core_parameters*

Specifies the parameters needed to configure the core component. It is mandatory. This command will fail if none of the core parameters are specified.

Examples

```
configure_core -component_name {PF_CCC_C0} -params "GL1_0_IS_USED:false"
"GL0_0_IS_USED:true" "GL0_0_OUT_FREQ:200"}
configure_core -component_name {Core_UART} -params {"BAUD_VAL_FRCTN_EN:false"
"RX_FIFO:0" "RX_LEGACY_MODE:0" "TX_FIFO:1" "USE_SOFT_FIFO:1"}
```

See Also

Tcl Command Documentation Conventions

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 | SPM_OTP
```

Supported tool_names

The following table lists the supported tool_names.

tool_name	Parameter (-params)	Description
"CONFIGURE_PROG_OPTIONS " on page 100	See the topic for parameter names and values.	See the topic for 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 100	See the topic for parameter names and values.	See the topic for description.

tool_name	Parameter (-params)	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 105	See the topic for parameter names and values.	See the topic for description.
SPM_OTP	See the topic for parameter names and values.	See the topic for description.

See Also

Tcl documentation conventions

create_and_configure_core

Tcl command; creates a configured core component for a core selected from the Libero Catalog.

To use this command to create a configured core component with valid parameters and values, it is recommended to use the GUI to configure the core as desired. Then export the core configuration Tcl description by selecting the "Export Component Description(Tcl)" action on the right-click menu of the component in the Design Hierarchy. You can then use the exported Tcl command to create the configured core in a regular Tcl script.

```
create_and_configure_core \
-core_vlnv Vendor:Library:Name:version \
-component_name component_name \
[-params core_parameters]
```

Arguments

-core_vlnv *Vendor:Library:Name:Version*

Specifies the version identifier of the core being configured. It is mandatory.

-component_name *component_name*

Specifies the name of the configured core component. It is mandatory.

-params *core_parameters*

Specifies the parameters that need to be configured for the core component. It is optional. If the core parameters are not specified with this argument, the component is configured and generated with the core's default configuration. It is recommended to specify all the core parameters of interest as a part of this argument in this command.

Examples

```
create_and_configure_core -core_vlnv {Actel:SgCore:PF_CCC:1.0.115} -
component_name {PF_CCC_C3} -params { \
"PLL_IN_FREQ_0:25" \
"GL0_0_IS_USED:true" \
"GL0_0_OUT_FREQ:150" \
"GL0_1_IS_USED:true" \
"GL0_1_OUT_FREQ:50" }
```

Notes

For DirectCore and Solutions cores, refer to the core handbook or the core user guide for a list of valid parameters and values.

See Also

[Tcl Command Documentation Conventions](#)

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

Value	Description
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 }
```

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]
```

create_smartdesign

Tcl command; creates a SmartDesign.

```
create_smartdesign \
-sd_name smartdesign_component_name
```

Arguments

`-sd_name` *smartdesign_component_name*

Specifies the name of the SmartDesign component to be created. It is mandatory.

Examples

```
create_smartdesign -sd_name {top}
```

See Also

Tcl Command Documentation Conventions

delete_component

Tcl command; deletes a component from the Design Hierarchy.

```
delete_component \
-component_name component_name
```

Arguments

`-component_name` *component_name*

Specifies the name of the component to be deleted. It is mandatory.

Examples

```
delete_component -component_name {component}
delete_component -component_name {shifter}
```

See Also

Tcl Command Documentation Conventions

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}
```

download_latest_cores

This Tcl command is used to download the latest cores into the vault. A project does not need to be open to run this command.

```
download_latest_cores
```

This command takes no arguments.

If there are no cores to be downloaded, you will see the following message:

```
Info:All the latest cores are present in the vault.
```

edit_profile

Tcl command; sets the same values as the Add or Edit Profile dialog box. See the online help for more information.

```
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_ba_files

Tcl command to export the backannotated files. The backannotated files are <design_name>_ba.v (Verilog backannotated netlist) or <design_name>_ba.vhd (VHDL backannotated netlist) and <design_name>_ba.sdf (Standard Delay Format) timing file. These files are passed to the default simulator for postlayout simulation.

```
export_ba_files
-export_dir {absolute path to folder location}
-export_file_name {name of file}
-vhdl {value}
-min_delay {value}
```

Arguments

`-export_dir` {*absolute path to directory/folder location*}

Folder/directory location.

`-export_file_name` {*name of file*}

File name to generate the files. If not specified, it takes <design_name> as the default.

`-vhdl` {*value*}

Generates the <design_name>_ba.v and <design_name>_ba.sdf when set to 0 and
<design_name>_ba.vhd and <design_name>_ba.sdf when set to 1. Default is 0.

-min_delay {*value*}

Set to 1 to export enhanced min delays to include your best-case timing results in your Back Annotated file.
Default is 0.

Returns

Returns 0 on success, 1 on failure.

Example

```
export_ba_files\
   -export_dir {E:\designs\export\sd1}\
   -export_file_name {test}\
   -vhdl 0\
   -min_delay 1
```

export_bitstream_file

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

```
 export_bitstream_file
[-file_name file]
[-export_dir dir]
[-format PPD | STP | DAT | SPI | HEX]
[-for_ihp 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]
[-zeroization_likenew_action 0 | 1]
[-zeroization_unrecoverable_action 0 | 1]
[-master_backlevel_bypass 0 | 1]
[-uek1_backlevel_bypass 0 | 1]
[-uek2_backlevel_bypass 0 | 1]
[-master_include_plaintext_passkey 0 | 1]
[-uek1_include_plaintext_passkey  0 | 1]
[-uek2_include_plaintext_passkey 0 | 1]
```

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 *PPD | STP | CHAIN_STP | DAT | SPI | HEX*

Specifies the bitstream file formats to be exported. Space is used as a delimiter. If omitted, PPD and DAT
files will be exported.

-for_ihp *0 | 1*

Specifies to export the bitstream files for Microsemi In House Programming(IHP).

Zeroization Options:

`-zeroization_likenew_action 0 | 1`

Specifies that all the data will be erased and the device can be reprogrammed immediately

`-zeroization_unrecoverable_action 0 | 1`

Specifies that all the data will be erased and the device cannot be reprogrammed and it must be scrapped.

Security-related options:

Note: One of the trusted_facility file or master_file or encrypted_uek1_file or encrypted_uek2_file must be set to "1". 1 indicates that this particular file type will be exported; 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 `uek1_file` and `uek2_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 only be 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 either SECURITY or SECURITY, FABRIC and SNVM

Notes:

1. The SECURITY option is available in -bitstream_file_components only when file type is MASTER in – bitstream_file_type.

2. SNVM should be programmed with FABRIC

3. Security only programming must be performed only on erased or new devices. If performed on device with fabric programmed, the fabric will be disabled after performing security only programming. You must reprogram the fabric to re-enable it.

`-encrypted_uek1_file` *0 | 1*

`-encrypted_uek1_file_components` *FABRIC | SNVM*

Specifies the components of the design that will be saved to uek1 bitstream.

Note: SNVM should be programmed with FABRIC

`-encrypted_uek2_file` *0 | 1*

`-encrypted_uek2_file_components` *FABRIC | SNVM*

Specifies the components of the design that will be saved to uek2 bitstream.

Note: SNVM should be programmed with FABRIC

`-master_include_plaintext_passkey` *0 | 1*

Specifies that the master file includes plaintext passkey. This argument is optional.

`-uek1_include_plaintext_passkey` *0 | 1*

Specifies that uek1 includes plaintext passkey. This argument is optional.

`-uek2_include_plaintext_passkey` *0 | 1*

Specifies that uek2 includes plaintext passkey. This argument is optional.

Bypass Back Level Protection Options:

`-master_backlevel_bypass 0 | 1`

Specifies the Bypass Back Level protection for Golden/Recovery bitstream if back level protection is enabled in _master file.

-uek1_backlevel_bypass 0 | 1

Specifies the Bypass Back Level Protection for Golden/Recovery bitstream if back level protection is enabled in _uek1 file.

-uek2_backlevel_bypass 0 | 1

Specifies the Bypass Back Level Protection for Golden/Recovery bitstream if back level protection is enabled in _uek2 file.

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.

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 SNVM}
```

Export bitstream file for design with custom security options

Export bitstreams to master, uek1 and uek2 encrypted files. Master file to include security, fabric and SNVM components and Export Pass Key in Plaintext, uek1 and uek2 encrypted files to include FABRIC and SNVM with Like new Zeroization option enabled.

```
export_bitstream_file\
  -file_name {fftousram_new} \
  -export_dir
  {X:\10_docs_review\pf2.2_sp1\Programming_sars\99412\clkint_fftousram_ac_latch_launch\des
  igner\fftousram_new\export} \
  -format {PPD DAT STP HEX} \
  -for_ihp 1 \
  -master_file 1 \
  -master_file_components {SECURITY FABRIC SNVM} \
  -encrypted_uek1_file 1 \
  -encrypted_uek1_file_components {FABRIC SNVM} \
  -encrypted_uek2_file 1 \
  -encrypted_uek2_file_components {FABRIC SNVM} \
  -trusted_facility_file 0 \
  -trusted_facility_file_components {} \
  -zeroization_likenew_action 1 \
  -zeroization_unrecoverable_action 0 \
  -master_backlevel_bypass 0 \
  -uek1_backlevel_bypass 0 \
  -uek2_backlevel_bypass 0
  -master_include_plaintext_passkey 1 \
  -uek1_include_plaintext_passkey 0 \
  -uek2_include_plaintext_passkey 0
```

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_component_to_tcl

Tcl command; exports the Tcl command for the selected component. The components can be SmartDesign components, configured cores and HDL+ cores.

```
export_component_to_tcl \
-component component_name \
[-library library_name] \
[-package package_name] \
-file file_path
```

Arguments

-component *component_name*

Specifies the name of the component for which the Tcl command is exported. It is mandatory.

-library *library_name*

Specifies the name of the library the component belongs to. It is optional.

-package *package_name*

Specifies the name of the package the HDL+core belongs to. It is optional.

-file *file_path*

Specifies the path where you wish to export the Tcl file. It is mandatory.

Example

```
export_component_to_tcl -component {pattern_gen_checker} -library {work} -package {} -
file {./pattern_gen_checker.tcl}
```

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_fp_pdc

Tcl command to export the Floorplanning Physical Design Constraint (*.pdc) File. The exported file has a *_fp.pdc file name extension.

```
export_fp_pdc
-file {absolute path and name of *_fp.pdc file}
-mode {PDC_PLACE | PDC_FULL_PLACEMENT}
```

Arguments

-file {absolute path and name of *_fp.pdc file}

Specifies the *_fp.pdc file.

-mode {PDC_PLACE | PDC_FULL_PLACEMENT}

Use PDC_PLACE to export user's floorplanning constraints, for example, fixed logic and regions.

Use PDC_FULL_PLACEMENT to export information about all of the physical design constraints (I/O constraints, I/O Banks, routing constraints, region constraints, global and local clocks).

Returns

Returns 0 on success, 1 on failure.

Example

```
export_fp_pdc\
  -file {E:/designs/export/sd1_fp.pdc}\
  -mode {PDC_FULL_PLACEMENT}
```

export_ibis_file

Tcl command to export the IBIS (Input/Output Buffer Information Specification) model report. The exported file has a *.ibs file name extension.

```
export_ibis_file
-file {absolute path and name of *.ibs file}
```

Arguments

-file {absolute path and name of *.ibs file}

Specifies the IBIS file to export.

Returns

Returns 0 on success, 1 on failure.

Example

```
export_ibis_file\
  -file {E:/designs/export/sd1.ibs}
```

export_io_pdc

Tcl command to export the I/O constraints Physical Design Constraint (*.pdc) File. The exported file has a *_io.pdc file name extension.

```
export_io_pdc
-file {absolute path and name of *_io.pdc file}
```

Arguments

-file {*absolute path and name of *_io.pdc file*}

Specifies the *_io.pdc file.

Returns

Returns 0 on success, 1 on failure.

Example

```
export_io_pdc\
  -file {E:/designs/export/sd1_io.pdc}
```

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. See the online help for more information.

```
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_prog_job

Tcl command; configures the parameters for the FlashPro Express programming job to be exported.

```
 export_prog_job
-job_file_name file
-export_dir dir
-bitstream_file_type TRUSTED_FACILITY | MASTER | UEK1 | UEK2
-bitstream_file_components SECURITY | FABRIC | SNVM
-zeroization_likenew_action 0 | 1
-zeroization_unrecoverable_action 0 | 1
-program_design 0 | 1
-program_spi_flash 0 | 1
-include_plaintext_passkey 0 | 1-design_bitstream_format PPD | STP
```

Arguments

-job_file_name *file*

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

-export_dir *dir*

Location where the job file will be saved; any folder can be specified. The default folder is the Libero export folder.

-bitstream_file_type *TRUSTED_FACILITY | MASTER | UEK1 | UEK2*

Bitstream file to be included in the programming job. Only one bitstream file can be included in a programming job.

-bitstream_file_components *SECURITY | FABRIC | SNVM*

The list of components to be included in the programming job. Components should be delimited by space.

bitstream_file_components can be any one of SECURITY or SECURITY, FABRIC and SNVM

Notes:

1. The SECURITY option is available in -bitstream_file_components only when file type is MASTER in –bitstream_file_type.

2. SNVM must always be programmed with FABRIC.

3. Security-only programming must be performed only on erased or new devices. If performed on a device with fabric programmed, the fabric will be disabled after performing security-only programming. You must reprogram the fabric to re-enable it.

-zeroization_likenew_action 0 | 1

Specifies that all data will be erased and the device can be reprogrammed immediately.

-zeroization_unrecoverable_action 0 | 1

Specifies that all data will be erased. The device cannot be reprogrammed and it must be scrapped.

-program_design *0 | 1*

Specifies to program the design. This argument is optional.

-program_spi_flash *0 | 1*

Specifies to program SPI Flash. This argument is optional.

-include_plaintext_passkey *0 | 1*

Specifies to include plaintext passkey. This argument is optional.

-design_bitstream_format *PPD | STP*

Specifies the Bitstream file format. If omitted, the bitstream file will be in PPD format.

Example

```
          export_prog_job \
```

```
-job_file_name {fftousram_new} \
-export_dir
{X:\10_docs_review\12.0_Release\102018\clkint_fftousram_ac_latch_launch\designer\fftousra
m_new\export} \
-bitstream_file_type {MASTER} \
-bitstream_file_components {SECURITY FABRIC SNVM} \
-zeroization_likenew_action 0 \
-zeroization_unrecoverable_action 0 \
-program_design 1 \
-program_spi_flash 0 \
-include_plaintext_passkey 0 \
-design_bitstream_format {PPD}
```

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_component

Tcl command; generates a SmartDesign or a core component.

```
generate_component \
-component_name component_name \
[-recursive 0|1]
```

Arguments

-component_name *component_name*

Specifies the name of the SmartDesign component or the core component to be generated. It is mandatory.

-recursive *0|1*

Specifies if a SmartDesign component needs to be generated recursively. It is optional. It is '0' by default and generates only the specified component. If set to '1', all the dependent components which are in ungenerated state will be generated along with the SmartDesign component. It is recommended to generate all components individually.

Examples

The following command generates SmartDesign "sd2" only.

```
generate_component -component_name {sd2}
```

The following command generates SmartDesign "TOP" and all its dependent components which are in ungenerated state.

```
generate_component -component_name {TOP} -recursive 1
```

See Also

[Tcl Command Documentation Conventions](#)

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.

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.

```
import_files

-smartgen_core {file}
-ccp {file}
-stimulus {file}
-hdl_source {file}

-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}
-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.

-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.

-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 –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 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)

Value	Description
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: • 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.

Example

```
#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}
```

See Also

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

open_smartdesign

Tcl command; opens a SmartdDesign. You must either open or create a SmartDesign before using any of the SmartDesign specific commands "sd_*".

```
open_smartdesign \
-sd_name smartdesign_component_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component to be opened. It is mandatory.

Examples

```
open_smartdesign -sd_name {top}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

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

Value	Description
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

Value	Description
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

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 | 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.
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*.
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.

tool_name	Parameter	Description
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](#)

save_smartdesign

Tcl command; saves all the changes made in a SmartDesign component.

```
save_smartdesign \
-sd_name smartdesign_component_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component to be saved. It is mandatory.

Examples

```
save_smartdesign -sd_name {top}
```

See Also

[Tcl Command Documentation Conventions](#)

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).
RESTRICTPROBEPINS:1	(For SmartFusion2, IGLOO2 and RTG4 only) Sets to 1 to reserve your pins for probing if you intend to debug using SmartDebug.

See Also

How to Derive Required Part Information from A "Part Number"

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
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

SmartDesign Tcl Commands

The SmartDesign Tcl commands can be used to create a design in the SmartDesign. You must either create or open a SmartDesign before you can use any of the SmartDesign commands - sd_* .

All SmartDesign Tcl commands are supported by the PolarFire family.

sd_add_pins_to_group

Tcl command; adds one or more pins to a pin group on an instance in a SmartDesign component.

```
sd_add_pins_to_group \
-sd_name smartdesign_component_name \
-instance_name instance_name \
-group_name group_name \
-pin_names pin_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the instance on which the pin group is present. It is mandatory.

-group_name *group_name*

Specifies the name of the group to add the pins to. It is mandatory.

-pin_names *pin_names*

Specifies the list of instance pins to be added to the pin group. It is mandatory.

Examples

```
sd_add_pins_to_group -sd_name {TOP} -instance_name
{COREAXI4INTERCONNECT_C0_0} -group_name {Group} -pin_names {ARESETN ACLK}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_clear_pin_attributes

Tcl command; clears all attributes on one or more pins/ports in a SmartDesign. Pin attributes include pin inversion, mark as unused and constant value settings.

```
sd_clear_pin_attributes \
-sd_name smartdesign_component_name \
-pin_names port_or_pin_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_names *port_or_pin_names*

Specifies the name of the port/pin for which all the attributes must be cleared. It is mandatory.

Examples

```
sd_clear_pin_attributes -sd_name {sd1} -pin_names {RAM1K18_0:A_DOUT_CLK}
sd_clear_pin_attributes -sd_name {top} -pin_names {CARRY_OUT}
```

Notes

This command will not work on multiple pins/ports in this release. Support for multiple pins/ports will be provided in the next Libero release. This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_configure_core_instance

Tcl command; configures the parameters of a core instance (Direct Instantiation) in a SmartDesign component. This command is typically used after instantiating a core from the catalog directly into a SmartDesign component (Direct Instantiation) without first creating a component for the core (using sd_instantiate_core). This command can configure multiple core parameters at a time.

```
sd_configure_core_instance \
-sd_name smartdesign_component_name \
-instance_name core_instance_name \
-params core_parameters \
[-validate_rules 0|1]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the core instance in the SmartDesign which needs to be configured. It is mandatory.

-params *core_parameters*

Specifies the parameters that need to be configured for the core instance. It is mandatory.

-validate_rules *0|1*

Validates the rules of the updated configuration. It is optional.

Examples

```
sd_configure_core_instance -sd_name {SD1} -instance_name {COREFIFO_0} -
params {"SYNC:0" "param2:value2" "param3:value3"} -validate_rules 0
```

See Also

Tcl Command Documentation Conventions

sd_connect_instance_pins_to_ports

Tcl command; connects all pins of an instance to new SmartDesign top level ports.

```
sd_connect_instance_pins_to_ports \
-sd_name smartdesign_component_name \
-instance_name instance_name
```

Arguments

`-sd_name` *`smartdesign_component_name`*

Specifies the name of the SmartDesign component. It is mandatory.

`-instance_name` *`instance_name`*

Specifies the instance name for which all the pins must be connected to top level ports. It is mandatory. The instance pins are connected to new top level ports created with the same instance pin names. If a top level port with the same name already exists, then the tool automatically creates a new port with name <port_name>_<index> (index is an automatically generated integer starting at 0 such that the port name is unique in the SmartDesign).

Examples

```
sd_connect_instance_pins_to_ports -sd_name {top} -instance_name
{CORESPI_C0_0}
sd_connect_instance_pins_to_ports -sd_name {top} -instance_name
{ddr_out_0}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_connect_net_to_pins

Tcl command; connects a list of SmartDesign top level ports and/or instance pins to a net.

```
sd_connect_net_to_pins \
-sd_name smartdesign_component_name \
-net_name net_name \-pin_names port_or_pin_names
```

Arguments

`-sd_name` *`smartdesign_component_name`*

Specifies the name of the SmartDesign component. It is mandatory.

`-net_name` *`net_name`*

Specifies the name of the net to be connected to pins/ports in the SmartDesign component. It is mandatory.

`-pin_names` *`port_or_pin_names`*

Specifies the name of the ports/pins to be connected to the net in the SmartDesign. It is mandatory. The command will fail if:

- The ports/pins do not exist.

- The ports/pins and the net being connected are of different range/size.

- There is more than one port/pin driving the net.

Examples

```
sd_connect_net_to_pins -sd_name {shifter} -net_name {ready_net} -pin_names {"READY"}
sd_connect_net_to_pins -sd_name {top} -net_name {clk_net} -pin_names {CLK
RAM64x12_0:R_CLK RAM64x12_0:W_CLK}
```

Notes

This command is not required to build a SmartDesign component. It is not exported when you select Libero **Project** - `Export Script File` or '**Export Component Description(Tcl)**' on a SmartDesign component. This command is typically used in conjunction with 'sd_create_*_net' command to connect two or more ports/pins to a net.

See Also

[Tcl Command Documentation Conventions](#)

sd_connect_pins_to_constant

Tcl command; connects SmartDesign top level output ports or input instance pins to constant values.

```
sd_connect_pins_to_constant \
-sd_name smartdesign_component_name \
-pin_names port_or_pin_names \
-value constant_value
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_names *port_or_pin_names*

Specifies the names of the top level output ports or the instance level input pins to be tied to constant values. It is mandatory. Bus pins/ports and pin/port slices can also be tied to constant values. This command will fail if the specified port/pin does not exist. The command will also fail if the assigned object is a port of direction IN/INOUT or a pin of direction OUT/INOUT.

-value *constant_value*

Specifies the constant value to be assigned to the port/pin. It is mandatory. The acceptable values to this argument are GND/VCC/hexadecimal numbers.

Examples

```
sd_connect_pins_to_constant -sd_name {top} -pin_name {bypass} -value
{GND}

sd_connect_pins_to_constant -sd_name {top} -pin_name {sle_0:en} -value
{VCC}

sd_connect_pins_to_constant -sd_name {top} -pin_name {ram64x12_0:w_data}
-value {0x7f}
```

Notes

This command will not work on multiple pins/ports in this release. Support for multiple pins/ports will be provided in the next Libero release.

See Also

[Tcl Command Documentation Conventions](#)

sd_connect_pin_to_port

Tcl command; connects a SmartDesign instance pin to a new top level port. This command is equivalent to the 'Promote to Top Level` GUI action on an instance pin.

```
sd_connect_pin_to_port \
-sd_name smartdesign_component_name \
```

```
 -pin_name pin_name \
[-port_name port_name]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_name *pin_name*

Specifies the name of the instance level pin that needs to be connected to a top level port. It is mandatory.

-port_name *port_name*

Specifies the name of the new top level port that the instance pin will be connected to. It is optional. If the port name is not specified, the new port takes the name of the instance pin. If the port name as defined by these rules already exists, the tool automatically creates a new port with name <port_name>_<index> (index is an automatically generated integer starting at 0 such that the port name is unique in the SmartDesign).

Examples

```
sd_connect_pin_to_port -sd_name {top} -pin_name {DFN1_0:D}
sd_connect_pin_to_port -sd_name {top} -pin_name {DFN1_0:Q} -port_name
{Q_OUT}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

[Tcl Command Documentation Conventions](#)

sd_connect_pins

Tcl command; connects a list of SmartDesign top level ports and/or instance pins together.

```
sd_connect_pins \
-sd_name smartdesign_component_name \
-pin_names port_or_pin_or_slice_names_
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_names *port_or_pin_or_slice_names*

Specifies the port names, pin names and/or slice names to be connected together. It is mandatory. This command will fail if the ports, pins or slices do not exist. This command will also fail if the ports, pins and/or slices are not of the same size/range.

Examples

```
sd_connect_pins -sd_name {top} -pin_names {CLK MACC_PA_0:CLK DFN1_0:CLK}
sd_connect_pins -sd_name {top} -pin_names {MACC_PA_0:A
RAM1K20_0:A_DIN[17:0]}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_create_bif_net

Tcl command; creates a bus interface (BIF) net in a SmartDesign component. Any net created must be connected to two or more ports/pins using the command "sd_connect_net_to_pins".

```
sd_create_bif_net \
-sd_name smartdesign_component_name \
-net_name net_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-net_name *net_name*

Specifies the name of the net to be added in the SmartDesign component. It is mandatory. The command will fail if there is an already existing net with the same name.

Examples

```
sd_create_bif_net -sd_name {TOP} -net_name {bifnet1}
```

Note: This new bif net is visible in the UI only when it is connected to two or more ports/pins using the command "sd_connect_net_to_pins" as shown below.

```
sd_connect_net_to_pins -sd_name {TOP} -net_name {bifnet1} -pin_names {"AHBmmaster0"
"CoreAHBLite_C0_0:AHBmmaster0"}
```

Notes

This command is not required to build a SmartDesign component. It is not exported when you select Libero **Project** - `**Export Script File**' or **'Export Component Description(Tcl)**' on a SmartDesign component. This command is used to manually create a Tcl script and specify a new name to the net that connects two or more ports/pins.

See Also

Tcl Command Documentation Conventions

sd_create_bif_port

Tcl command; creates a SmartDesign Bus Interface port of a given type. This command is used to create top level Bus Interface ports in a SmartDesign component to connect to the instance level Bus Interface ports of the same type.

To use this command, it is recommended to first use the GUI to instantiate the core component or the HDL module with Bus Interface port to be promoted in the SmartDesign. Then use the UI action "Promote to Top Level" on the Bus Interface port of interest and export the Tcl script for the SmartDesign component by selecting " Export Component Description(Tcl)" on the right-click menu of the SmartDesign component in the Design Hierarchy. You can then use the Tcl command 'sd_create_bif_port' from the exported Tcl script (note to change the SmartDesign name in the command) to create a bus interface port anywhere in a regular Libero script. Note that there can be different Bus Interface types and roles defined by the arguments –port_bif_vlnv and – port_bif_role.

```
sd_create_bif_port \
-sd_name smartdesign_component_name \
-port_name port_name \
-port_bif_vlnv vendor:library:name:version \
-port_bif_role port_bif_role \
-port_bif_mapping [bif_port_name:port_name]+
```

Arguments

`-sd_name` *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-port_name` *port_name*

Specifies the name of the Bus Interface port to be added in the SmartDesign. It is mandatory.

`-port_bif_vlnv {vendor:library:name:version}`

Specifies the version identifier of the Bus Interface port to be added in the SmartDesign. It is mandatory.

`-port_bif_role {port_bif_role}`

Specifies the role of the Bus Interface port to be added in the SmartDesign. Role values depend on the type of Bus Interface (VLNV) that is being defined for the port. The figure below shows the roles for different Bus Interface ports supported by Libero.

Name	Vendor	Library	Role
AHB	AMBA	AMBA2	master
AHB	AMBA	AMBA2	slave
AHB	AMBA	AMBA2	mirroredMaster
AHB	AMBA	AMBA2	mirroredSlave
APB	AMBA	AMBA2	master
APB	AMBA	AMBA2	slave
APB	AMBA	AMBA2	mirroredMaster
APB	AMBA	AMBA2	mirroredSlave
AXI	AMBA	AMBA3	master
AXI	AMBA	AMBA3	slave
AXI	AMBA	AMBA3	mirroredMaster
AXI	AMBA	AMBA3	mirroredSlave
AXI	AMBA	AMBA3	system
AXI4	AMBA	AMBA4	master
AXI4	AMBA	AMBA4	slave
AXI4	AMBA	AMBA4	mirroredMaster
AXI4	AMBA	AMBA4	mirroredSlave
DDR3	Actel	busdef.memory	master
DDR3	Actel	busdef.memory	slave
PF_APB_LINK	Actel	busdef.link	master
PF_APB_LINK	Actel	busdef.link	slave
PF_CDR_CLK	Actel	busdef.clock	master
PF_CDR_CLK	Actel	busdef.clock	slave
PF_DRI	Actel	busdef.dri	master
PF_DRI	Actel	busdef.dri	slave
PF_DRI	Actel	busdef.dri	mirroredMaster
PF_DRI	Actel	busdef.dri	mirroredSlave
PF_TXPLL_XCVR_CLK	Actel	busdef.clock	master
PF_TXPLL_XCVR_CLK	Actel	busdef.clock	slave

`-port_bif_mapping {[bif_port_name:port_name ]+}`

Specifies the mapping between the bus interface formal names and the SmartDesign ports mapped onto that bus interface port. It is mandatory.

Examples

```
sd_create_bif_port -sd_name {sd1} -port_name {BIF_1} -port_bif_vlnv
{AMBA:AMBA2:APB:r0p0} -port_bif_role {slave} -port_bif_mapping {\
"PADDR:PADDR" \
"PSELx:pselx" \
"PENABLE:PENABLE" \
```

```
            "PWRITE:PWRITE" \
            "PRDATA:PRDATA" \
            "PWDATA:PWDATA" \
            "PREADY:PREADY" \
            "PSLVERR:PSLVERR" }
```

See Also

Tcl Command Documentation Conventions

sd_create_bus_net

Tcl command; creates a bus net of a given range in a SmartDesign component. Any net created must be connected to two or more ports/pins using the command "sd_connect_net_to_pins".

```
sd_create_bus_net \
-sd_name smartdesign_component_name \
-net_name net_name \
-net_range [left_index_range:right_index_range]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-net_name *net_name*

Specifies the name of the net to be added in the SmartDesign component. It is mandatory.

-net_range *[left_index_range:right_index_range]*

Specifies the range of the net added to the SmartDesign component. The range is defined by its left and right range indices. It is mandatory.

Examples

```
sd_create_bus_net -sd_name {top} -net_name {ab1} -net_range {[5:0]}
```

Note: This new net is visible in the UI only when it is connected to two or more ports/pins using the command "sd_connect_net_to_pins" as shown below.

```
sd_connect_net_to_pins -sd_name {top} -net_name {ab1} -pin_names {a RAM64x12_0:R_ADDR}
```

Notes

This command is not required to build a SmartDesign component. It is not exported when you select Libero **Project** - `**Export Script File**' or '**Export Component Description(Tcl)**' on a SmartDesign component. This command is used to manually create a Tcl script and specify a new name to the net that connects two or more ports/pins.

See Also

Tcl Command Documentation Conventions

sd_create_bus_port

Tcl command; creates a bus port of a given range in a SmartDesign component.

```
sd_create_bus_port \
-sd_name smartdesign_component_name \
```

```
-port_name port_name \-port_direction IN|OUT|INOUT \
-port_range {[left_range_index:right_range_index]}
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-port_name *port_name*

Specifies the name of the bus port added to be SmartDesign component. It is mandatory.

-port_direction *IN|OUT|INOUT*

Specifies the direction of the bus port added to the SmartDesign component. It is mandatory.

-port_range *{[left_range_index:right_range_index]}*

Specifies the range of the bus port added to the SmartDesign component. The range is defined by the left and right indices. It is mandatory. The range must be specified inside the square brackets.

Examples

```
sd_create_bus_port -sd_name {top} -port_name {test_port13} -port_direction {OUT} -
port_range {[9:36]}
sd_create_bus_port -sd_name {top} -port_name {test_port4} -port_direction {IN} -
port_range {[31:0]}
```

See Also

Tcl Command Documentation Conventions

sd_create_pin_group

Tcl command; creates a group of pins in a SmartDesign component. A pin group is only used to manage the complexity of the SmartDesign canvas. There is no actual netlist functionality related to pin group commands. Pin groups cannot be created for top level ports.

```
sd_create_pin_group \
-sd_name smartdesign_component_name \
-instance_name instance_name \
[-group_name group_name] \
[-pin_names pin_to_be_added_to_the_group]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the instance on which the pin group is added. It is mandatory.

-group_name *group_name*

Specifies the name of the pin group. It is optional. If the group name is not specified, the default name will be 'Group'. If the name 'Group' is already taken, then the group name will be 'Group_<index>' (index is auto-incremented).

-pin_names *pins_to_be_added_to_the_group*

Specifies the list of instance pins to be added to the pin group. It is optional.

Examples

```
sd_create_pin_group -sd_name {TOP} -instance_name
{COREAXI4INTERCONNECT_C0_0} -group_name {MyGroup} -pin_names {ACLK
ARESETN}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_create_pin_slices

Tcl command; creates slices for a SmartDesign top level bus port or an instance level bus pin.

```
sd_create_pin_slices \
-sd_name smartdesign_component_name \
-pin_name port_or_pin_name \
-pin_slices port_or_pin_slices
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_name *port_or_pin_name*

Specifies the name of the bus port or bus pin to be sliced. It is mandatory. This command will fail if the port/pin is scalar or if the bus port/pin does not exist.

-pin_slices *port_or_pin_slices*

Specifies the port/pin slices as a list of bus ranges which must be contained within the port/pin bus range. It is mandatory. This command will fail if the sliced object is top level OUT/INOUT port and the slice ranges overlap. This command will also fail if the sliced object is an instance level IN/INOUT pin and the slice ranges overlap.

Examples

```
sd_create_pin_slices -sd_name {sub} -pin_name {Rdata} -pin_slices {[4:3] [2:0]} # top
level port slicing

sd_create_pin_slices -sd_name {sub} -pin_name {DDR_memory_arbiter_C0_0:VIDEO_RDATA_4_O}
-pin_slices {[3:3] [2:0]} # instance level pin slicing
```

See Also

[Tcl Command Documentation Conventions](#)

sd_create_scalar_net

Tcl command; creates a scalar net in a SmartDesign component. Any net created must be connected to two or more ports/pins using the command "sd_connect_net_to_pins".

```
sd_create_scalar_net \
-sd_name smartdesign_component_name \
-net_name net_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-net_name *net_name*

Specifies the name of the net added to the SmartDesign component. It is mandatory.

Examples

```
sd_create_scalar_net -sd_name {top} -net_name {clk_net}
```

Note: This new net is visible in the UI only when it is connected to two or more ports/pins using the command "sd_connect_net_to_pins" as shown below

```
sd_connect_net_to_pins -sd_name {top} -net_name {clk_net} -pin_names {CLK
RAM64x12_0:R_CLK RAM64x12_0:W_CLK}
```

Notes

This command is not required to build a SmartDesign component. It is not exported when you select Libero **Project** - `**Export Script File**' or **'Export Component Description(Tcl)**' on a SmartDesign component. This command is used to manually create a Tcl script and specify a new name to the net that connects two or more ports/pins.

See Also

[Tcl Command Documentation Conventions](#)

sd_create_scalar_port

Tcl command; creates a scalar port in a SmartDesign component.

```
sd_create_scalar_port \
-sd_name smartdesign_component_name \
-port_name port_name \
-port_direction IN|OUT|INOUT
```

Arguments

-sd_name *smartdesign_component_name*
Specifies the name of the SmartDesign component. It is mandatory.

-port_name *port_name*
Specifies the name of the port added to the SmartDesign component. It is mandatory.

-port_direction *IN|OUT|INOUT*
Specifies the direction of the port added to the SmartDesign component. It is mandatory.

Examples

```
sd_create_scalar_port -sd_name {main} -port_name {po2} -port_direction {INOUT}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_delete_instances

Tcl command; deletes one or more instances from a SmartDesign component.

```
sd_delete_instances \
-sd_name smartdesign_component_name \
-instance_names instance_names
```

Arguments

-sd_name *smartdesign_component_name*
Specifies the name of the SmartDesign component. It is mandatory.

-instance_names *instance_names*
Specifies the instance names to be deleted. It is mandatory.

Examples

```
sd_delete_instances -sd_name {top} -instance_names {RAM64X12_0}
```

```
sd_delete_instances -sd_name {SUB} -instance_names {coreahblite_c0_0
coreriscv_axi4_c0_0 pf_ccc_c0_0}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_delete_nets

Tcl command; deletes one or more nets from the SmartDesign component.

```
sd_delete_nets \
-sd_name smartdesign_component_name \
-net_names net_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-net_names *net_names*

Specifies the net names to be deleted. It is mandatory.

Examples

```
sd_delete_nets -sd_name {topp} -net_names {B_REN_0}
```

Notes

This command will not delete multiple nets in this release. Support for deleting multiple nets will be provided in the next Libero release. This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_delete_pin_group

Tcl command; deletes a pin group from an instance in a SmartDesign component.

```
sd_delete_pin_group \
-sd_name smartdesign_component_name \
-instance_name instance_name \
-group_name group_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-group_name *group_name*

Specifies the name of the pin group to be deleted. It is mandatory.

-instance_name *instance_name*

Specifies the name of the instance from which the group pin needs to be deleted. It is mandatory.

Examples

```
sd_delete_pin_group -sd_name {TOP} -instance_name
{COREAXI4INTERCONNECT_C0_0} -group_name {Group}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_delete_pin_slices

Tcl command; deletes SmartDesign top level port slices or instance pin slices.

```
sd_create_pin_slices \
-sd_name smartdesign_component_name \
-pin_name port_or_pin_name \
-pin_slices port_or_pin_slices
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_name *port_or_pin_name*

Specifies the name of the bus port or bus pin for which the slices must be deleted. It is mandatory.

-pin_slices *port_or_pin_slices*

Specifies the ranges of the port and/or pin slices to be deleted. It is mandatory.

Examples

```
sd_delete_pin_slices -sd_name {top} -pin_name {MACC_pa_0:p} -pin_slices
{[21] [13] [28]} # deletes instance pin slices
sd_delete_pin_slices -sd_name {top} -pin_name {A} -pin_slices {[17:16]
[15:1] [0]} # deletes top level port slices
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

[Tcl Command Documentation Conventions](#)

sd_delete_ports

Tcl command; deletes one or more ports from the SmartDesign component.

```
sd_delete_ports \
-sd_name smartdesign_component_name \
-port_names port_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-port_names *port_names*

Specifies the names of the ports to be deleted. It is mandatory.

Examples

sd_delete_ports -sd_name {sd1} -port_names {REF_CLK_0}

Notes

This command will not work on multiple ports in this release. Support for multiple ports will be provided in the next Libero release. This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_disconnect_instance

Tcl command; clears all the connections on an instance in a SmartDesign component.

```
sd_disconnect_instance \
-sd_name smartdesign_component_name \
-instance_name instance_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the instance for which all the connections must be cleared. It is mandatory.

Examples

sd_disconnect_instance -sd_name {sd1} -instance_name {RAM1K18_1}

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_disconnect_pins

Tcl command; disconnects a list of SmartDesign top level ports and/or instance pins from the net they are connected to.

```
sd_disconnect_pins \
-sd_name smartdesign_component_name \
-pin_names port_or_pin_or_slice_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-pin_names` *port_or_pin_or_slice_names*

Specifies the port, pin and/or slice names to be disconnected. It is mandatory. This command will fail if the ports, pins and/or slices do not exist.

Examples

```
sd_disconnect_pins -sd_name {topp} -pin_names {B_ren
RAM1K20_0:B_ADRR[12]}
sd_disconnect_pins -sd_name {SD1} -pin_names {AND2_0:B AND3_0:B AND3_0:A
PF_XCVR_ERM_C0_0:LANE0_RX_READY}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

[Tcl Command Documentation Conventions](#)

sd_duplicate_instance

Tcl command; creates a new instance in a SmartDesign with the same module/component as the original instance.

```
sd_duplicate_instance \
-sd_name smartdesign_component_name \
-instance_name instance_name \[-duplicate_instance_name duplicate_instance_name]
```

Arguments

`-sd_name` *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-instance_name` *instance_name*

Specifies the name of the instance to be duplicated. It is mandatory.

`-duplicate_instance_name` *duplicate_instance_name*

Specifies the name of the duplicate instance. It is optional. If the duplicate_instance_name is not specified, it will be automatically generated as <instance_name><index> (index is an automatically generated integer starting at 0 such that the instance name is unique in the SmartDesign).

Examples

```
sd_duplicate_instance -sd_name {top} -instance_name {PF_CCC_C0_0}
sd_duplicate_instance -sd_name {top} -instance_name {SUB_0} -
duplicate_instance_name {T1}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

[Tcl Command Documentation Conventions](#)

sd_hide_bif_pins

Tcl command; hides one or more already exposed internal scalar or bus pins/ports of a Bus Interface pin/port.

```
sd_hide_bif_pins \
-sd_name smartdesign_component_name \
-bif_pin_name name_of_the_bif_pin_or_port \-pin_names pins_or_ports_to_be_exposed
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-bif_pin_name *name_of_the_bif_pin_name*

Specifies the name of the Bus Interface pin for which the internal pins must be hidden. It is mandatory.

-pin_name *pins_to_be_exposed*

Specifies the bus interface internal pin/port names to be hidden. It is mandatory.

Examples

```
sd_hide_bif_pins -sd_name {sd1} -bif_pin_name {COREAXI4INTERCONNECT_C0_0:AXI4mmaster0} -
pin_names {COREAXI4INTERCONNECT_C0_0:MASTER0_AWADDR}
```

```
sd_hide_bif_pins -sd_name {SD1} -bif_pin_name {CLKS_FROM_TXPLL_0} -pin_names
{TX_PLL_LOCK_0}
```

Notes

This command will not hide multiple pins/ports in this release. Support to hide multiple pins/ports will be provided in the next Libero release. This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_instantiate_component

Tcl command; instantiates a Libero SmartDesign component or a core component into another SmartDesign component.

```
sd_instantiate_component \
-sd_name smartdesign_component_name \
-component_name component_module_name \
[-instance_name instance name]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component in which other components will be instantiated. It is mandatory.

-component_name *component_module_name*

Specifies the name of the component being instantiated in the SmartDesign component. It is mandatory. The components include SmartDesign components, core components created for different types of cores from the catalog and blocks.

-instance_name *instance name*

Specifies the instance name of the Libero component being instantiated in the SmartDesign component. It is optional. By default, the instance name is <component_module_name>_<index> (index is an automatically generated integer starting at 0 such that the instance name is unique in the SmartDesign).

Examples

```
sd_instantiate_component -sd_name {sub} -component_name {sd1} -
instance_name {sd1_0}
sd_instantiate_component -sd_name {top} -component_name {PF_CCC_C0}
```

See Also

Tcl Command Documentation Conventions

sd_instantiate_core

Tcl command; instantiates a core from the catalog directly into a SmartDesign component (Direct Instantiation) without first having to create a component for the core. The file-set related to the core is generated only when the SmartDesign in which the core is instantiated is generated. The GUI equivalent of this command is not currently supported in Libero. To instantiate a core in a SmartDesign component in the GUI, you have to first create a component for the core.

```
sd_instantiate_core \
-sd_name smartdesign_component_name \-core_vlnv vendor:library:name:verison \[-instance_name
instance_name]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-core_vlnv *vendor:library:name:verison*

Specifies the version identifier of the core being instantiated in the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the instance name of the core being instantiated in the SmartDesign. It is optional. By default, the instance name is <core_name>_<index> (index is an automatically generated integer starting at 0 such that the instance name is unique in the SmartDesign).

Examples

```
sd_instantiate_core -sd_name {top} -core_vlnv
{Actel:DirectCore:COREAXI4INTERCONNECT:2.5.100} -instance_name
{COREAXI4INTERCONNECT_C0_0}
```

See Also

Tcl Command Documentation Conventions

sd_instantiate_hdl_core

Tcl command; instantiates a HDL+ core in a SmartDesign component. HDL+ core definition must be created on a HDL module before using this command.

```
sd_instantiate_hdl_core \
-sd_name smartdesign_component_name \
-hdl_core_name hdl_core_module_name \
[-instance_name instance_name]
```

Arguments

`-sd_name` *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-hdl_core_name` *hdl_core_module_name*

Specifies the name of the HDL+ core module being instantiated in the SmartDesign component. It is mandatory.

`-instance_name` *instance_name*

Specifies the instance name of the HDL+ core being instantiated in the SmartDesign. It is optional. By default, the instance name is <hdl_core_module_name>_<index> (index is an automatically generated integer starting at 0 such that the instance name is unique in the SmartDesign).

Examples

```
sd_instantiate_hdl_core -sd_name {top} -hdl_core_name {temp} -instance_name {temp3}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_instantiate_hdl_module

Tcl command; instantiates a HDL module in a SmartDesign component. The HDL file in which the HDL module is defined must be imported/linked before running this command.

```
sd_instantiate_hdl_module \
-sd_name smartdesign_component_name \-hdl_module_name hdl_module_name \-hdl_file hdl_file \
[-instance_name instance_name]
```

Arguments

`-sd_name` *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-hdl_module_name` *hdl_module_name*

Specifies the name of the HDL module being instantiated in the SmartDesign component. It is mandatory.

`-hdl_file` *hdl_file*

Specifies the path of the HDL file in which the HDL module is defined. The HDL file path can be relative to project folder for imported files but the path has to be complete for linked files. It is mandatory.

`-instance_name` *instance_name*

Specifies the instance name of the HDL module. It is optional. By default, the instance name is <hdl_module_name>_<index> (index is an automatically generated integer starting at 0 such that the instance name is unique in the SmartDesign).

Examples

```
sd_instantiate_hdl_module -sd_name {top} -hdl_module_name {and1} -hdl_file {hdl\and1.v}
sd_instantiate_hdl_module -sd_name {top} -hdl_module_name {and_ex} -hdl_file
{hdl\and_ex.v} -instance_name {test_hdl_hdl_module_name_plus1_1}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_instantiate_macro

Tcl command; instantiates a Microsemi primitive macro in a SmartDesign component.

```
sd_instantiate_macro \
-sd_name smartdesign_component_name \
-macro_name macro_module_name |
[-instance_name instance_name]
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-macro_name *macro_module_name*

Specifies the name of the macro being instantiated in the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the instance name of the macro. It is optional. By default, the instance name is <macro name>_<index> (index is an automatically generated integer starting at 0 such that the instance name is unique in the SmartDesign).

Examples

```
sd_instantiate_macro -sd_name {TOP} -macro_name {MX2} -instance_name {MX2_0}
```
$$sd\_instantiate\_macro \ \text{-sd\_name} \ \{TOP\} \ \text{-macro\_name} \ \{MACC\_PA\}$$

See Also

[Tcl Command Documentation Conventions](#)

sd_invert_pins

Tcl command; inverts one or more top level ports or instance level pins in a SmartDesign.

```
sd_invert_pins \
-sd_name smartdesign_component_name \
-pin_names port_or_pin_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_names *port_or_pin_names*

Specifies the port or pin names to be inverted. It is mandatory. This parameter can take multiple values. This command will fail if the port/pin does not exist.

Examples

```
sd_invert_pins -sd_name {main} -pin_names {A}
sd_invert_pins -sd_name {main} -pin_names {MX2_1:S MX2_1:Y A B}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_mark_pins_unused

Tcl command; marks one or more SmartDesign instance level output pins as unused. When an output pin is marked as unused, no Design Rule Check (DRC) warning will be printed for floating output pins while generating the SmartDesign.

```
sd_mark_pins_unused \
-sd_name smartdesign_component_name \
-pin_names port_or_pin_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-pin_names *port_or_pin_names*

Specifies the names of the instance pins to be marked as unused. It is mandatory.

Examples

```
sd_mark_pins_unused -sd_name {top} -pin_names {PF_CCC_C0_0:PLL_LOCK_0}
```

Notes

This command will not work on multiple pins in this release. Support for multiple pins will be provided in the next Libero release.

See Also

Tcl Command Documentation Conventions

sd_remove_pins_from_group

Tcl command; removes one or more pins from a pin group on an instance in a SmartDesign.

```
sd_remove_pins_from_group \
-sd_name smartdesign_component_name \
-instance_name instance_name \
-group_name group_name \
-pin_names pin_names
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the instance on which the pin group is present. It is mandatory.

-group_name *group_name*

Specifies the name of the pin group from which pins need to be removed. It is mandatory.

-pin_names *pin_names*

Specifies the list of pin names to be removed from the pin group. It is mandatory.

Examples

```
sd_remove_pins_from_group -sd_name {TOP} -instance_name
{COREAXI4INTERCONNECT_C0_0} -group_name {Group} -pin_names {ARESETN ACLK}
```

See Also

Tcl Command Documentation Conventions

sd_rename_instance

Tcl command; renames an instance in a SmartDesign component. This command can be used to rename any type of instances (instances of other SmartDesigns components, core components, HDL modules, HDL+ cores and Microsemi macros) in a SmartDesign.

```
sd_rename_instance \
-sd_name component_name \
-current_instance_name instance_name \
-new_instance_name new_instance_name
```

Arguments

-sd_name *component_name*

Specifies the name of the SmartDesign component in which the instance name has to be renamed. It is mandatory.

-current_instance_name *instance_name*

Specifies the name of the instance to be renamed. It is mandatory.

-new_instance_name *new_instance_name*

Specifies the new instance name. It is mandatory.

Examples

```
sd_rename_instance -sd_name {top} -current_instance_name {DFN1_0} -
new_instance_name {DFN1_new}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

[Tcl Command Documentation Conventions](#)

sd_rename_net

Tcl command; renames a net in a SmartDesign component.

```
sd_rename_net \
-sd_name smartdesign_component_name \
-current_net_name current_net_name \-new_net_name new_net_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-current_net_name *current_net_name*

Specifies the name of the net to be renamed in the SmartDesign. It is mandatory.

-new_net_name *new_net_name*

Specifies the new name of the net in the SmartDesign. It is mandatory.

Examples

```
sd_rename_net -sd_name {top} -current_net_name {clk_net} -new_net_name {clk_rclk_wclk}
sd_rename_net -sd_name {PCIe_EP_Demo} -current_net_name {USER_RESETN} -new_net_name
{reset_input}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

sd_rename_pin_group

Tcl command; renames a pin group on an instance in a SmartDesign component.

```
sd_rename_pin_group \
-sd_name smartdesign_component_name \
-instance_name instance_name \
-current_group_name current_pin_group_name \
-new_pin_group_name new_pin_group_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the instance on which the pin group is present. It is mandatory.

-current_group_name *current_pin_group_name*

Specifies the name of the pin group to be renamed. It is mandatory.

-new_group_name *new_group_name*

Specifies the new name of the pin group. It is mandatory.

Examples

```
sd_rename_pin_group -sd_name {TOP} -instance_name
{COREAXI4INTERCONNECT_C0_0} -current_group_name {Group} -new_group_name
{MyNewGroup}
```

See Also

Tcl Command Documentation Conventions

sd_rename_port

Tcl command; renames a SmartDesign port.

```
sd_rename_port \
-sd_name smartdesign_component_name \
-current_port_name port_name \
-new_port_name new_port_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-current_port_name *port_name*

Specifies the name of the port to be renamed in the SmartDesign component. It is mandatory. Note that only port names can be renamed, and not port types (scalar ports cannot be renamed as bus ports and vice versa).

-new_port_name *new_port_name*

Specifies the new name of the specified port. It is mandatory.

Examples

```
sd_rename_port -sd_name {top} -library {work} -current_port_name {c1} -new_port_name
{c2}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

[Tcl Command Documentation Conventions](#)

sd_save_core_instance_config

Tcl command; this command is used to save the core instance configuration specified using one or more 'sd_configure_core_instance' commands. This command is typically used after configuring a core instance in a SmartDesign, to save that core instance's configuration.

```
sd_save_core_instance_config \
-sd_name smartdesign_component_name \
-instance_name core_instance_name
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

-instance_name *instance_name*

Specifies the name of the core instance in the SmartDesign for which the configuration must be saved. It is mandatory.

Examples

```
sd_save_core_instance_config -sd_name {SD1} -instance_name {COREFIFO_0}
```

See Also

[Tcl Command Documentation Conventions](#)

sd_show_bif_pins

Tcl command; exposes one or more internal scalar or bus pins/ports of a Bus Interface pin/port. A Bus Interface pin/port is usually a group of normal scalar or bus pins/ports grouped together and used to connect instances that have similar interfaces. The internal pins/ports underneath the Bus Interface pin/port may have to be exposed in some cases to connect to some logic in the design.

```
sd_show_bif_pins \
-sd_name smartdesign_component_name \
-bif_pin_name name_of_the_bif_pin_or_port \
-pin_names pins_or_ports_to_be_exposed
```

Arguments

-sd_name *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-bif_pin_name` *name_of_the_bif_pin_or_port*

Specifies the name of the Bus Interface pin/port for which the internal pins/ports need to be exposed. It is mandatory.

`-pin_names` *pins_or_ports_to_be_exposed*

Specifies the names of the Bus Interface internal pins/ports to be exposed. It is mandatory.

Examples

```
sd_show_bif_pins -sd_name {TOP} -bif_pin_name {COREAXI4INTERCONNECT_C0_0:AXI4mmaster0} -
pin_names {COREAXI4INTERCONNECT_C0_0:MASTER0_AWADDR}
```

```
sd_show_bif_pins -sd_name {SD1} -bif_pin_name {CLKS_FROM_TXPLL_0} -pin_names
{TX_PLL_LOCK_0}
```

Notes

This command will not expose multiple pins/ports in this release. Support to expose multiple scalar or bus pins/ports will be provided in the next Libero release.

See Also

Tcl Command Documentation Conventions

sd_update_instance

Tcl command; updates an instance in a SmartDesign with its latest definition. This command is useful when the interface (port-list) of the component/module instantiated in a SmartDesign has changed. This command can be used to update any type of instance such as instances of other SmartDesign components, core components, HDL modules and HDL+ cores in a SmartDesign.

```
sd_update_instance \
-sd_name smartdesign_component_name \
-instance_name instance_name
```

Arguments

`-sd_name` *smartdesign_component_name*

Specifies the name of the SmartDesign component. It is mandatory.

`-instance_name` *instance_name*

Specifies the name of the instance to be updated. It is mandatory.

Examples

```
sd_update_instance -sd_name {top} -instance_name {CORESMIP_C0_0}
```

Notes

This command is not required to build a SmartDesign component. This command maps to an interactive user action in the SmartDesign Canvas and will not be present in the exported SmartDesign component Tcl description.

See Also

Tcl Command Documentation Conventions

HDL Core Tcl Commands

create_hdl_core

This Tcl command is used to create a core component from an HDL core.

```
create_hdl_core \
-module {module_name} \
-file {file_path}
-library {library_name} \
-package {package_name}
```

Arguments

-module *{module_name}*

Specify the module name for which you want to create a core component. This is a mandatory argument.

-file *{file_path}*

Specify the file path of the module from which you create a core component. This is a mandatory argument.

-library *{library_name}*

Specify the library name from which you want to create a HDL core. This is an optional argument.

-package *{package_name}*

Specify the package name from which you want to create a core component. This is an optional argument.

Example

```
create_hdl_core -file {./HDL_CORE_TEST/hdl/hdl_core.v} -module {test_hdl_core}
```

See Also

Tcl Command Documentation Conventions

hdl_core_add_bif

This Tcl command adds a bus interface to an HDL core.

```
hdl_core_add_bif \
-hdl_core_name {hdl_core_name} \
-bif_definition {Name:Vendor:Library:Role} \
-bif_name {bus_interface_name} \
[-signal_map {signal_map}]
```

Arguments

-module *{module_name}*

Specify the HDL core name to which the bus interface needs to be added. This is a mandatory argument.

-bif_definition *{Name:Vendor:Library:Role}*

Specify the Bus Interface Definition Name, Vendor, Library and Bus Role of the core in the format {N:V:L:R}. This is a mandatory argument.

-bif_name *{bus_interface_name}*

Specify the bus interface port name being added to the HDL core. This is a mandatory argument.

```
-signal_map {signal_map}
```
This argument is used to specify the signal map of the bus interface. This is an optional argument.

Example

```
hdl_core_add_bif -hdl_core_name {test_hdl_core} -bif_definition {AHB:AMBA:AMBA2:master} -
bif_name {BIF_1}
```

See Also

Tcl Command Documentation Conventions

hdl_core_assign_bif_signal

Maps a bus interface signal definition name to an HDL core module port name.

```
hdl_core_assign_bif_signal
-hdl_core_name {hdl_core_name} \
-bif_name {bus_interface_name} \
-bif_signal_name {bif_signal_name} \
-core_signal_name {core_signal_name}
```

Arguments

```
-hdl_core_name {hdl_core_name}
```
Specify the HDL core name to which the bus interface signal needs to be added. This is a mandatory argument.

```
-bif_name {bus_interface_name}
```
Specify the bus interface name for which you want to map a core signal. This is a mandatory argument.

```
-bif_signal_name {bus_interface_signal_name}
```
Specify the bus interface signal name that you want to map with the core signal name. This is a mandatory argument.

```
-core_signal_name {core_signal_name}
```
Specify the core signal name for which you want to map the bus interface signal name. This is a mandatory argument.

Example

```
hdl_core_assign_bif_signal -hdl_core_name {test_hdl_core} -bif_name {BIF_1} -
bif_signal_name {HWRITE} -core_signal_name {myHRESULT}
```

See Also

Tcl Command Documentation Conventions

hdl_core_delete_parameters

This Tcl command deletes parameters from a HDL core definition.

```
hdl_core_delete_parameters
-hdl_core_name {module_name} \
-parameters {parameter_list}
```

Arguments

```
-hdl_core_name {hdl_core_name}
```
Specify the HDL core name from which you want to delete parameters. This is a mandatory argument.

```
-parameters {parameter_list}
```

Specify the list of parameters from a HDL core. This is typically done to remove parameters from the list of parameters that was automatically extracted using the hdl_core_extract_ports_and_params command. This is a mandatory argument.

Example

```
hdl_core_delete_parameters -hdl_core_name {test_hdl_core} -parameters {WIDTH}
```

See Also

[Tcl Command Documentation Conventions](#)

hdl_core_extract_ports_and_parameters

This Tcl command automatically extracts ports and generic parameters from an HDL core module description.

```
hdl_core_extract_ports_and_parameters \
-hdl_core_name {hdl_core_name}
```

Arguments

-hdl_core_name *hdl_core_name*}

Specifies the HDL core name from which you want to extract signal names and generic parameters. This is a mandatory argument.

Example

```
hdl_core_extract_ports_and_params -hdl_core_name {test_hdl_core}
```

See Also

[Tcl Command Documentation Conventions](#)

hdl_core_remove_bif

Remove an existing bus interface from an HDL core.

```
hdl_core_remove_bif \
-hdl_core_name {hdl_core_name} \
-bif_name {bus_interface_name}
```

Arguments

-module {*module_name*}

Specify the HDL core name from which the bus interface needs to be removed. This is a mandatory argument.

-bif_name {*bus_interface_name*}

Specify the bus interface name that needs to be removed from the HDL core. This is a mandatory argument.

Example

```
hdl_core_remove_bif -hdl_core_name {mod1} -bif_name {BIF_1}
```

See Also

[Tcl Command Documentation Conventions](#)

hdl_core_rename_bif

Rename an existing bus interface port of a HDL core.

```
hdl_core_rename_bif
-hdl_core_name {hdl_core_name} \
-current_bif_name {current_bus_interface_name} \
-new_bif_name {new_bus_interface_name}
```

Arguments

-hdl_core_name {*hdl_core_name*}

Specify the HDL core name for which the bus interface needs to be renamed. This is a mandatory argument.

-current_bif_name {*current_bus_interface_name*}

Specify the bus old bus interface name that needs to be renamed for the HDL core. This is a mandatory argument.

-new_bif_name {*new_bus_interface_name*}

Specify the new bus interface name that needs to be updated for the HDL core. This is a mandatory argument.

Example

```
hdl_core_rename_bif -hdl_core_name {test_hdl_plus} -current_bif_name {BIF_2} -
new_bif_name {BIF_3}
```

See Also

[Tcl Command Documentation Conventions](#)

hdl_core_unassign_bif_signal

Unmap an existing bus interface signal from a bus interface.

```
hdl_core_unassign_bif_signal
-hdl_core_name {hdl_core_name} \
-bif_name {bus_interface_name} \
-bif_signal_name {bif_signal_name}
```

Arguments

-hdl_core_name {*hdl_core_name*}

Specify the HDL core name from which the bus interface signal needs to be deleted. This is a mandatory argument.

-bif_name {*bus_interface_name*}

Specify the bus interface name for which you want to unassign a core signal. This is a mandatory argument.

-bif_signal_name {*bus_interface_signal_name*}

Specify the bus interface signal name for which you want to unassign a core signal. This argument is mandatory.

Example

```
hdl_core_unassign_bif_signal -hdl_core_name {test_hdl_plus} -bif_name {BIF_2} -
bif_signal_name {PENABLE}
```

See Also

[Tcl Command Documentation Conventions](#)

remove_hdl_core

This Tcl command removes an HDL core component from the current project.

```
remove_hdl_core \
-hdl_core_name {hdl_core_name}
```

Arguments

-hdl_core_name {*hdl_core_name*}

Specify the module name from which you want to delete a core component. This is a mandatory argument.

Example

```
remove_hdl_core -hdl_core_name {test_hdl_core}
```

See Also

[Tcl Command Documentation Conventions](#)

Command Tools

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 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 specific Tcl commands to configure JTAG chain. For details on JTAG chain programming Tcl commands, refer to the Tcl commands section in the Libero SoC Online Help.

Do not include any 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 script. If, for example, the 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.

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
program_fabric	true	false	Include fabric component in the programming bitstream.

Name	Value	Description
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.

IO_PROGRAMMING_STATE

IO_PROGRAMMING_STATE is a command tool used in the configure_tool Tcl command. The configure_tool -name {IO_PROGRAMMING_STATE } Tcl command loads the I/O State information from a file during programming. The file used for loading the I/O State informatio during programming is specified in a parameter to the command. Refer to the Specify I/O States During Programming Dialog Box for details .

```
configure_tool -name {IO_PROGRAMMING_STATE} -params\
{ios_file:absolute_path_to_i/o_state_information_file}
```

Example

```
configure_tool -name {IO_PROGRAMMING_STATE} -params\
  {ios_file:d:\sd_prj\tony_sf2\designer\sd1\sd1.ios}
```

Return

Returns 0 on success and 1 on failure.

PLACEROUTE

To place and route a design in Libero SoC, you must first configure the PLACEROUTE tool with the configure_tool command and then execute the PLACEROUTE tool with the run_tool command.

configure_tool

```
configure_tool -name {PLACEROUTE} [-params {[name:value ]+}]+
```

Parameters

Name	Value	Description
TDPR	true | false | 1 | 0	Set to true or 1 to enable Timing-Driven Place and Route. Default is 1.
PDPR	true | false | 1 | 0	Set to true or 1 to enable Power-Driven Place and Route. Default is false or 0.

Name	Value	Description
IOREG_COMBINING	true | false | 1 | 0	Set to true or 1 to enable I/O Register Combining. Default is false or 0.
GB_DEMOTION	true | false | 1 | 0	Set to true or 1 to enable Global Pins Demotion. Default is true or 1
REPLICATION	true | false | 1 | 0	Set to true or 1 to enable Driver Replication. Default is false or 0
EFFORT_LEVEL	true | false | 1 | 0	Set to true or 1 to enable High Effort Layout to optimize design performance. Default is false or 0.
INCRPLACEANDROUTE	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	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	1-25	Specifies the number of passes to run. The default is 5. Maximum is 25.
START_SEED_INDEX	1-100	Indicates the random seed index which is 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	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_CIOCK: Use the slowest clock frequency in the design in a given pass as the performance reference for the layout pass. • 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	Clock_Name	Applies only when MULTI_PASS_CRITERIA is set to SPECIFIC_CLOCK. It specifies the name of

Name	Value	Description
		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	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.
RGB_COUNT	1-18	Allows an entity to override the placer's RGB/RCLK bandwidth constraint. This option is useful for Block Creation.

Return Value

Returns 0 on success and 1 on failure.

run_tool

run_tool -name {PLACEROUTE}

Parameters

None

Return Value

Returns 0 on success and 1 on failure.

Example

```
configure_tool -name {PLACEROUTE}\
  -params {EFFORT_LEVEL:true}\
  -params {GB_DEMOTION:true}\
  -params {INCRPLACEANDROUTE:false}\
  -params {IOREG_COMBINING:false}\
  -params {MULTI_PASS_CRITERIA:VIOLATIONS}\
  -params {MULTI_PASS_LAYOUT:false}\
  -params {NUM_MULTI_PASSES:5}\
  -params {PDPR:false}\
  -params {REPAIR_MIN_DELAY:true}\
  -params {REPLICATION:false}\
  -params {SLACK_CRITERIA:WORST_SLACK}\
  -params {SPECIFIC_CLOCK:}\
  -params {START_SEED_INDEX:1}\
  -params {STOP_ON_FIRST_PASS:false}\
  -params {TDPR:true}\
  -params {USE_RAM_MATH_INTERFACE_LOGIC:false}
run_tool -name{PLACEROUTE}
```

PROGRAMDEVICE

PROGRAMDEVICE 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 program the design in Libero SoC, you must first configure the PROGRAMDEVICE tool with configure_tool command and then execute the PROGRAMDEVICE command with the run_tool command.

Use the commands to configure your programming action and the programming procedures associated with the program action.

```
configure_tool -name {PROGRAMDEVICE}
-params {prog_action:params_value}
-params {prog_optional_procedures:params_value}
-params {skip_recommended_procedures:params_value}
```

```
run_tool -name {PROGRAMDEVICE}
```

configure_tool –name {PROGRAMDEVICE} parameter:value pair

Name	Value	Description
prog_action	String { PROGRAM | | VERIFY | ERASE | DEVICE_INFO |	PROGRAM – Programs all selected family features: FPGA Array, targeted eNVM clients and security settings.

Name	Value	Description
	READ_IDCODE | ENC_DATA_AUTHENTICATION | VERIFY_DIGEST}	VERIFY – Verifies all selected family features: FPGA Array, targeted eNVM clients and security settings. ERASE – Erases the selected family features: FPGA Array and security settings. DEVICE_INFO – Displays the IDCODE, the design name, the checksum, and device security settings and programming environment information programmed into the device.READ_IDCODE – Reads the device ID code from the device. ENC_DATA_AUTHENTICATION - Encrypted bitstream authentication data. VERIFY_DIGEST – Calculates the digests for the components included in the bitstream and compares them against the programmed values
prog_optional_procedures	Depends on the action from the prog_action parameter.	This parameter is optional. It is only required when the user wants to enable optional procedure.
skip_recommended_procedures	Depends on the action from the prog_action parameter.	This parameter is optional. It is used to deselect recommended procedures.

run_tool –name {PROGRAMDEVICE} Parameter:value pair

Name	Value	Description
None		

Example

```
configure_tool \
    -name {PROGRAMDEVICE} \
    -params {prog_action:VERIFY_DIGEST} \
    -params {prog_optional_procedures:DO_ENABLE_USER_PUBLIC_KEY } \
    -params {skip_recommended_procedures:DO_ENABLE_FABRIC DO_ENABLE_SNVM }
configure_tool -name {PROGRAMDEVICE} -params {prog_action:DEVICE_INFO}
run_tool –name {PROGRAMDEVICE} #Takes no parameters
```

Return

`configure_tool -name {PROGRAMDEVICE}` returns 0 on success and 1 on failure.

`run_tool -name {PROGRAMDEVICE}` returns 0 on success and 1 on failure.

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}
```

PROGRAMMER_INFO

PROGRAMMER_INFO is a command tool used in configure_tool. Configure_tool -name {PROGRAMMER_INFO} sets the programmer settings, similar to the way FlashPro commands set the programmer settings. This command supports FlashPro3, FlashPro4, FlashPro5, and FlashPro6.

```
configure_tool -name {PROGRAMMER_INFO}
-params [{name:value}]
```

The following tables list the parameter names and values.

configure_tool –name {PROGRAMMER_INFO} Parameter:value (FlashPro6)

Name	Value	Description
Flashpro6_force_freq	String {OFF | ON}	For FlashPro6 Programmer only.
Flashpro6_ freq	Integer (Hertz)	For FlashPro6 Programmer only.

configure_tool –name {PROGRAMMER_INFO} Parameter:value (FlashPro5)

Name	Value	Description
flashpro5_clk_mode	String {free_running_clk | discrete_clocking}	For FlashPro5 Programmer only.
flashpro5_force_freq	String {OFF | ON}	For FlashPro5 Programmer only.
flashpro5_ freq	Integer (Hertz)	For FlashPro5 Programmer only.
flashpro5_vpump	String {ON | OFF}	For FlashPro5 Programmer only.

configure_tool –name {PROGRAMMER_INFO} Parameter:value (FlashPro4)

Name	Value	Description
flashpro4_clk_mode	String {free_running_clk | discrete_clocking}	For FlashPro4 Programmer only.
flashpro4_force_freq	String {OFF | ON}	For FlashPro4 Programmer only.
flashpro4_ freq	Integer (Hertz)	For FlashPro4 Programmer only.
flashpro4_vpump	String {ON | OFF}	For FlashPro4 Programmer only.

configure_tool –name {PROGRAMMER_INFO} parameter:value (FlashPro3)

Name	Value	Description
flashpro3_clk_mode	String {free_running_clk | discrete_clocking}	For FlashPro3 Programmer only.
flashpro3_force_freq	String {OFF | ON}	For FlashPro3 Programmer only.
flashpro3_ freq	Integer (Hertz)	For FlashPro3 Programmer only.
flashpro3_vpump	String {ON | OFF}	For Flash For FlashPro3 Programmer only.

For a detailed description of the parameters and values, refer to Programmer Settings in the Libero Online Help.

Examples

For FlashPro3 programmer

```
configure_tool –name {PROGRAMMER_INFO}\
  -params {flashpro3_clk_mode:free_running_clk}\
  -params {flashpro3_force_freq:OFF}\
  -params {flashpro3_freq:400000}\
  -params {flashpro3_vpump:ON}
```

For FlashPro4 programmer

```
configure_tool –name {PROGRAMMER_INFO}\
  -params {flashpro4_clk_mode:free_running_clk\}
  -params {flashpro4_force_freq:OFF}\
  -params {flashpro4_freq:400000}\
  -params {flashpro4_vpump:ON}
```

For FlashPro5 programmer

```
configure_tool –name {PROGRAMMER_INFO}\
  -params {flashpro5_clk_mode:free_running_clk}\
  -params {flashpro5_force_freq:OFF}\
  -params {flashpro5_freq:400000}\
  -params {flashpro5_vpump:ON}
```

For FlashPro6 programmer

```
configure_tool –name {PROGRAMMER_INFO}\
  -params {flashpro6_force_freq:OFF}\
  -params {flashpro6_freq:400000}
```

Return

Returns 0 on success and 1 on failure.

SPM

To configure security using Tcl, you must use the configure_tool Tcl command to pass the SPM configuration parameters.

```
configure_tool -name {SPM}
-params {name:value}
[-params {name:value}]+
```

configure_tool –name {SPM} parameter:value pair

Note: true | 1 will select the checkbox in the SPM UI

Name	Type	Value	Description
back_level_protection	bool	false | true | 1 | 0	If true, will set back level protection; Update Policy
back_level_update_version	Integer	0 - 65535	Set back level version; Update Policy
debug_passkey	hex	64 hex characters	Value of DPK; Debug Policy
disable_authenticate_action	bool	false | true | 1 | 0	Disables Authenticate action
disable_autoprog_iap_services	bool	false | true | 1 | 0	Disables Auto Programming and IAP Services
disable_debug_jtag_boundary_scan	bool	false | true | 1 | 0	Disables debug JTAG Boundary Scan
disable_debug_read_temp_volt	bool	false | true | 1 | 0	Disables reading temperature and voltage sensor (JTAG/SPI Slave)
disable_debug_ujtag	bool	false | true | 1 | 0	Disables debug UJTAG
disable_ext_zeroization	bool	false | true | 1 | 0	Disables external zeroization through JTAG/SPI Slave
disable_external_digest_check	bool	false | true | 1 | 0	Disables external Fabric/sNVM digest requests through JTAG/SPI Slave
disable_jtag	bool	false | true | 1 | 0	Disables JTAG
disable_program_action	bool	false | true | 1 | 0	Disables Program action
disable_puf_emulation	bool	false | true | 1 | 0	Disables external access to PUF emulation through JTAG/SPI Slave
disable_smartdebug_debug	bool	false | true | 1 | 0	Disables user debug access and active probes
disable_smartdebug_live_probe	bool	false | true | 1 | 0	Disables SmartDebug Live Probe
disable_smartdebug_snvm	bool	false | true | 1 | 0	Disables SmartDebug sNVM
disable_spi_slave	bool	false | true | 1 | 0	Disables SPI Slave interface

Name	Type	Value	Description
disable_user_encryption_key_1	bool	false | true | 1 | 0	If true, disables UEK1; Key Mode Policy
disable_user_encryption_key_2	bool	false | true | 1 | 0	If true, disables UEK2; Key Mode Policy
disable_user_encryption_key_3	bool	false | true | 1 | 0	If true, disables UEK3 Note: UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices. All other devices will set this to false by default. See the SmartFusion2 SoC FPGA and IGLOO2 FPGA Security Best Practices User Guide for details. Key Mode Policy
disable_verify_action	bool	false | true | 1 | 0	Disables Verify action
fabric_update_protection	string	open | disabled	Fabric update protection. Open – updates allowed using user defined encryption keys. Disabled – disables Erase/Write operations.
security_factory_access	string	open | disabled	Microsemi factory test mode access. Open – factory test mode access allowed. Disabled – disables factory test mode access.
security_key_mode	string	custom | default	Default – bitstream encryption with default key. Custom – custom security options
snvm_update_protection	string	open | disabled	sNVM update protection. Open – updates allowed using user defined encryption keys. Disabled – disables Write operations.
user_encryption_key_1	hex	64 hex characters	Value of UEK1
user_encryption_key_2	hex	64 hex characters	Value of UEK2
user_passkey_1	hex	64 hex characters	Value of Flashlock/UPK1
user_passkey_2	hex	64 hex characters	Value of UPK2

Example

```
configure_tool \
```

```
        -name {SPM} \
        -params {back_level_protection:false} \
        -params {back_level_update_version: 32} \
        -params {disable_smartdebug_live_probe:false} \
        -params {disable_smartdebug_snvm:false} \
        -params {disable_user_encryption_key_1:false} \
        -params {disable_user_encryption_key_2:false} \
```

Return

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.
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.
AUTO_COMPILE_POINT	Boolean {true | false | 1 | 0}	Set to true or 1 to enable Automatic Compile Point during synthesis. Set to false or 0 to disable Automatic Compile Point during synthesis. The default is 0 or false.
SYNPLIFY_OPTIONS	String	Specifies additional synthesis-specific options. Options specified by this parameter override

Name	Value	Description
		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|DISCAR D}	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_C ONFLICTS	String {ERROR|KEEP|LOCK|DISCAR D}	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_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_RCLK INT_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.

Name	Value	Description
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.

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 {AUTO_COMPILE_POINT:true} -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.

VERIFYPOWER

VERIFYPOWER is a command tool used in run_tool. The command run_tool passes a script file that contains power-specific Tcl commands to the VERIFYPOWER command and executes it.

```
run_tool -name {VERIFYPOWER} –script {power_analysis.tcl}
```

where

<power_analysis.tcl> is a script that contains power-specific Tcl commands. You can include power-specific Tcl commands to generate power reports. See the sample power_analysis Tcl Script below for details.

Return

Returns 0 on success and 1 on failure.

Example

```
run_tool -name {VERIFYPOWER} -script {<power_analysis.tcl>}
```

Sample power_analysis Tcl Script <power_analysis.tcl>

The following example changes SmartPower operating condition settings from the default to 40C junction temperature and 1.25V VDD.

It then creates a report called A4P5000_uSRAM_POWER_64X18_power_report.txt.

\# Change from pre-defined temperature and voltage mode (COM,IND,MIL) to SmartPower custom

```
smartpower_set_temperature_opcond -use "design"
smartpower_set_voltage_opcond -voltage "VDD" -use "design"
```

\# Set the custom temperature to 40C ambient temperature.

```
smartpower_temperature_opcond_set_design_wide -typical 40 -best 40 -worst 40
```

\# Set the custom voltage to 1.25V

```
smartpower_voltage_opcond_set_design_wide -voltage "VDD" -typical 1.25 -best 1.25 -worst
1.25
```

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}
#Run Post-synthesis simulation
run_tool -name {SIM_POSTSYNTH}
```

Return Value

Returns 0 on success and 1 on failure.

SmartTime Tcl Commands

create_clock

Tcl command; creates a clock constraint on the specified ports/pins, or a virtual clock if no source other than a name is specified.

```
create_clock [-name clock_name] [-add] -period period_value
[-waveform> edge_list][source_objects]
```

Arguments

-period *period_value*

Specifies the clock period in nanoseconds. The value you specify is the minimum time over which the clock waveform repeats. The period_value must be greater than zero.

-name *clock_name*

Specifies the name of the clock constraint. You must specify either a clock name or a source.

-add

Specifies that a new clock constraint is created at the same source port as the existing clock without overriding the existing constraint. The name of the new clock constraint with the -add option must be different than the existing clock constraint. Otherwise, it will override the existing constraint, even with the -add option. The -name option must be specified with the -add option.

-waveform *edge_list*

Specifies the rise and fall times of the clock waveform in ns over a complete clock period. There must be exactly two transitions in the list, a rising transition followed by a falling transition. You can define a clock starting with a falling edge by providing an edge list where fall time is less than rise time. If you do not specify -waveform option, the tool creates a default waveform, with a rising edge at instant 0.0 ns and a falling edge at instant (period_value/2)ns.

source_objects

Specifies the source of the clock constraint. The source can be ports, pins, or nets in the design. If you specify a clock constraint on a pin that already has a clock, the new clock replaces the existing one. You must specify either a source or a clock name.

Description

Creates a clock in the current design at the declared source and defines its period and waveform. The static timing analysis tool uses this information to propagate the waveform across the clock network to the clock pins of all sequential elements driven by this clock source.

The clock information is also used to compute the slacks in the specified clock domain that drive optimization tools such as place-and-route.

Examples

The following example creates two clocks, one on port CK1 with a period of 6, and the other on port CK2 with a period of 6, a rising edge at 0, and a falling edge at 3:

```
create_clock -name {my_user_clock} -period 6 CK1
create_clock -name {my_other_user_clock} -period 6 -waveform {0 3} {CK2}
```

The following example creates a clock on port CK3 with a period of 7, a rising edge at 2, and a falling edge at 4:

```
create_clock -period 7 -waveform {2 4} [get_ports {CK3}]
```

The following example creates a new clock constraint clk2, in addition to clk1, on the same source port clk1 without overriding it.

```
create_clock -name clk1 -period 10 -waveform {0 5} [get_ports clk1]
create_clock -name clk2 -add -period 20 -waveform {0 10} [get_ports clk1]
```

The following example does not add a new clock constraint, even with the -add option, but overrides the existing clock constraint because of the same clock names. Note: To add a new clock constraint in addition to the existing clock constraint on the same source port, the clock names must be different.

```
create_clock -name clk1 -period 10 -waveform {0 5} [get_ports clk1]
create_clock -name clk1 -add -period 50 -waveform {0 25} [get_ports clk1]
```

See Also

create_generated_clock

Tcl Command Documentation Conventions

create_generated_clock

Tcl command; creates an internally generated clock constraint on the ports/pins and defines its characteristics.

```
create_generated_clock [-name clock_name] [-add] [-master_clock clock_name] -source
reference_pin [-divide_by divide_factor] [-multiply_by multiply_factor] [-invert] source[-edges
values] [-edge_shift values]
```

Arguments

-name *clock_name*

Specifies the name of the clock constraint.

-add

Specifies that the generated clock constraint is a new clock constraint in addition to the existing one at the same source. The name of the clock constraint should be different from the existing clock constraint. With this option, -master_clock option and -name options must be specified.

-master_clock *clock_name*

Specifies the master clock used for the generated clock when multiple clocks fan into the master pin. This option must be used in conjunction with -add option of the generated clock.

Notes:

1. The master_clock option is used only with the -add option for the generated clocks.
2. If there are multiple master clocks fanning into the same reference pin, the first generated clock specified will always use the first master clock as its source clock.
3. The subsequent generated clocks specified with the -add option can choose any of the master clocks as their source clock (including the first master clock specified).

-source *reference_pin*

Specifies the reference pin in the design from which the clock waveform is to be derived.

-divide_by *divide_factor*

Specifies the frequency division factor. For instance if the *divide_factor* is equal to 2, the generated clock period is twice the reference clock period.

-multiply_by *multiply_factor*

Specifies the frequency multiplication factor. For instance if the *multiply_factor* is equal to 2, the generated clock period is half the reference clock period.

-invert

Specifies that the generated clock waveform is inverted with respect to the reference clock.

source

Specifies the source of the clock constraint on internal pins of the design. If you specify a clock constraint on a pin that already has a clock, the new clock replaces the existing clock. Only one source is accepted. Wildcards are accepted as long as the resolution shows one pin.

`-edges` *values*

Specify the integer values that represent the edges from the source clock that form the edges of the generated clock. Three values must be specified to generate the clock. If you specify less than three, a tool tip indicates an error.

`-edge_shift` *values*

Specify a list of three floating point numbers that represents the amount of shift, in nanoseconds, that the specified edges are to undergo to yield the final generated clock waveform. These floating point values can be positive or negative. Positive value indicates a shift later in time, while negative indicates a shift earlier in time.

For example: An edge shift of {1 1 1} on the LSB generated clock, would shift each derived edge by 1 nanosecond.

To create a 200MHz clock from a 100MHz clock, use edge { 1 2 3} and edge shift {0 -2.5 -5.0}.

Description

Creates a generated clock in the current design at a declared source by defining its frequency with respect to the frequency at the reference pin. The static timing analysis tool uses this information to compute and propagate its waveform across the clock network to the clock pins of all sequential elements driven by this source.

The generated clock information is also used to compute the slacks in the specified clock domain that drive optimization tools such as place-and-route.

Examples

The following example creates a generated clock on pin U1/reg1:Q with a period twice as long as the period at the reference port CLK.

```
create_generated_clock -name {my_user_clock} -divide_by 2 -source [get_ports
{CLK}]  U1/reg1:Q
```

The following example creates a generated clock at the primary output of myPLL with a period ¾ of the period at the reference pin clk.

```
create_generated_clock -divide_by 3 -multiply_by 4  -source clk [get_pins {myPLL:CLK1}]
```

The following example creates a new generated clock gen2 in addition to gen1 derived from same master clock as the existing generated clock, and the new constraint is added to pin r1/CLK.

```
create_generated_clock -name gen1 -multiply_by 1 -source [get_ports clk1] [get_pins
r1/CLK]

create_generated_clock -name gen2 -add -master_clock clk1 -source [get_ports clk1] -
multiply_by 2 [get_pins r1/CLK]
```

The following example does not create a new generated clock constraint in addition to the existing clock, but will override even with the -add option enabled, because the same names are used.

```
create_generated_clock -name gen2 -source [get_ports clk1] -multiply_by 3 [get_pins
r1/CLK]

create_generated_clock -name gen2 -source [get_ports clk1] -multiply_by 4 -master_clock
clk1 -add [get_pins r1/CLK]
```

See Also

create_clock

Tcl Command Documentation Conventions

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|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`

Specifies the type of the report to be generated. It is mandatory.

Value	Description
timing	Timing Report
violations	Timing Violation Report
datasheet	Datasheet Report
bottleneck	Bottleneck Report
constraints_coverage	Constraints Coverage Report
combinational_loops	Combinational Loops Report

-analysis

Specifies the type of Analysis(Max Delay or Min Delay) Performed to generate the reports. It is optional.

Note: This argument should not be used to generate datasheet reports. The command may fail if this argument is used to generate datasheet report.

Value	Description
max	Timing report considers maximum analysis (default).
min	Timing report considers minimum analysis.

-format

Specifies the format in which the report is generated. It is optional.

Value	Description
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 of the generated report. It is mandatory.

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.

Parameter/Value	Description
-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.
-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.

Parameter/Value	Description
-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 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 maximum delays and does not filter paths based on slack threshold. It reports two paths per section and one expanded path per section.

```
report \
  -type violations \
  -analysis max \
  -use_slack_threshold no \
  -limit_max_paths yes \
  -max_paths 2 \
  -max_expanded_paths 1 \
  timing_viol.txt
```

The following example generates a datasheet report named datasheet.csv in CSV format.

```
report \
  -type datasheet \
  -format csv \
  datasheet.csv
```

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_clock_latency

Tcl command; defines the delay between an external clock source and the definition pin of a clock within SmartTime.

```
set_clock_latency -source [-rise][-fall][-early][-late] delay clock
```

Arguments

-source

Specifies the source latency on a clock pin, potentially only on certain edges of the clock.

-rise

Specifies the edge for which this constraint will apply. If neither or both rise are passed, the same latency is applied to both edges.

-fall

Specifies the edge for which this constraint will apply. If neither or both rise are passed, the same latency is applied to both edges.

-invert

Specifies that the generated clock waveform is inverted with respect to the reference clock.

-late

Optional. Specifies that the latency is late bound on the latency. The appropriate bound is used to provide the most pessimistic timing scenario. However, if the value of "-late" is less than the value of "-early", optimistic timing takes place which could result in incorrect analysis. If neither or both "-early" and "-late" are provided, the same latency is used for both bounds, which results in the latency having no effect for single clock domain setup and hold checks.

-early

Optional. Specifies that the latency is early bound on the latency. The appropriate bound is used to provide the most pessimistic timing scenario. However, if the value of "-late" is less than the value of "-early", optimistic timing takes place which could result in incorrect analysis. If neither or both "-early" and "-late" are provided, the same latency is used for both bounds, which results in the latency having no effect for single clock domain setup and hold checks.

delay

Specifies the latency value for the constraint.

clock

Specifies the clock to which the constraint is applied. This clock must be constrained.

Description

Clock source latency defines the delay between an external clock source and the definition pin of a clock within SmartTime. It behaves much like an input delay constraint. You can specify both an "early" delay and a"late" delay for this latency, providing an uncertainty which SmartTime propagates through its calculations. Rising and falling edges of the same clock can have different latencies. If only one value is provided for the clock source latency, it is taken as the exact latency value, for both rising and falling edges.

Examples

The following example sets an early clock source latency of 0.4 on the rising edge of main_clock. It also sets a clock source latency of 1.2, for both the early and late values of the falling edge of main_clock. The late value for the clock source latency for the falling edge of main_clock remains undefined.

```
set_clock_latency -source -rise -early 0.4 { main_clock  }
set_clock_latency -source -fall    1.2 { main_clock }
```

See Also

create_clock

create_generated_clock

Tcl Command Documentation Conventions

set_false_path

Tcl command; identifies paths that are considered false and excluded from the timing analysis in the current timing scenario.

```
set_false_path [-ignore_errors] [-from from_list] [-through through_list] [-to to_list]
```

Arguments

-ignore_errors

Specifies to avoid reporting errors for derived constraints targeting the logic that becomes invalid due to logic optimization. It is an optional argument. Some IPs may have extra logic present depending on other IPs used in the design but the synthesis tool will remove this logic if fewer IPs were used. In such cases, the implementation flow will halt without -ignore_errors flag.

Note: It is not recommended to use this flag outside similar use cases.

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-through *through_list*

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Description

The set_false_path command identifies specific timing paths as being false. The false timing paths are paths that do not propagate logic level changes. This constraint removes timing requirements on these false paths so that they are not considered during the timing analysis. The path starting points are the input ports or register clock pins, and the path ending points are the register data pins or output ports. This constraint disables setup and hold checking for the specified paths.

The false path information always takes precedence over multiple cycle path information and overrides maximum delay constraints. If more than one object is specified within one -through option, the path can pass through any objects.

You must specify at least one of the `-from`, `-to`, or `-through` arguments for this constraint to be valid.

Examples

The following example specifies all paths from clock pins of the registers in clock domain clk1 to data pins of a specific register in clock domain clk2 as false paths:

```
set_false_path -from [get_clocks {clk1}] -to reg_2:D
```

The following example specifies all paths through the pin U0/U1:Y to be false:

```
set_false_path -through U0/U1:Y
```

The following example specifies a derived false path constraint through the pin PCIe_Demo_0/SYSRESET_POR/POWER_ON_RESET_N

```
set_false_path -ignore_errors -through [ get_pins
{PCIe_Demo_0/SYSRESET_POR/POWER_ON_RESET_N } ]
```

See Also

[Tcl Command Documentation Conventions](#)

set_input_delay

Tcl command; creates an input delay on a port list by defining the arrival time of an input relative to a clock in the current scenario.

```
set_input_delay delay_value -clock clock_ref [-max] [-min] [-clock_fall] [-rise] [-fall] [-add_delay]
input_list
```

Arguments

delay_value

Specifies the arrival time in nanoseconds that represents the amount of time for which the signal is available at the specified input after a clock edge.

`-clock` *clock_ref*

Specifies the clock reference to which the specified input delay is related. This is a mandatory argument. If you do not specify -max or -min options, the tool assumes the maximum and minimum input delays to be equal.

`-max`

Specifies that delay_value refers to the longest path arriving at the specified input. If you do not specify -max or -min options, the tool assumes maximum and minimum input delays to be equal.

`-min`

Specifies that delay_value refers to the shortest path arriving at the specified input. If you do not specify -max or -min options, the tool assumes maximum and minimum input delays to be equal.

`-clock_fall`

Specifies that the delay is relative to the falling edge of the clock reference. The default is the rising edge.

`-rise`

Specifies that the delay is relative to a rising transition on the specified port(s). If -rise or -fall is not specified, then rising and falling delays are assumed to be equal.

`-fall`

Specifies that the delay is relative to a falling transition on the specified port(s). If -rise or -fall is not specified, then rising and falling delays are assumed to be equal.

`-add_delay`

Specifies that this input delay constraint should be added to an existing constraint on the same port(s). The -add_delay option is used to capture information on multiple paths with different clocks or clock edges leading to the same input port(s).

input_list

Provides a list of input ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Description

The set_input_delay command sets input path delays on input ports relative to a clock edge. This usually represents a combinational path delay from the clock pin of a register external to the current design. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool adds input delay to path delay for paths starting at primary inputs.

A clock is a singleton that represents the name of a defined clock constraint. This can be:

- a single port name used as source for a clock constraint
- a single pin name used as source for a clock constraint; for instance reg1:CLK. This name can be hierarchical (for instance toplevel/block1/reg2:CLK)
- an object accessor that will refer to one clock: [get_clocks {clk}]

Examples

The following example sets an input delay of 1.2ns for port data1 relative to the rising edge of CLK1:

```
set_input_delay 1.2 -clock [get_clocks CLK1] [get_ports data1]
```

The following example sets a different maximum and minimum input delay for port IN1 relative to the falling edge of CLK2:

```
set_input_delay 1.0 -clock_fall -clock CLK2 -min {IN1}
set_input_delay 1.4 -clock_fall -clock CLK2 -max {IN1}
```

The following example demonstrates an override condition of two constraints. The first constraint is overridden because the second constraint specifies a different clock for the same input:

```
set_input_delay 1.0 -clock CLK1 -max {IN1}
set_input_delay 1.4 -clock CLK2 -max {IN1}
```

The next example is almost the same as the previous one, however, in this case, the user has specified -add_delay, so both constraints will be honored:

```
set_input_delay 1.0 -clock CLK1 -max {IN1}
set_input_delay 1.4 -add_delay -clock CLK2 -max {IN1}
```

The following example is more complex:

- All constraints are for an input to port PAD1 relative to a rising edge clock CLK2. Each combination of {-rise, -fall} x {-max, -min} generates an independent constraint. But the max rise delay of 5 and the max rise delay of 7 interfere with each other.
- For a -max option, the maximum value overrides all lower values. Thus the first constraint will be overridden and the max rise delay of 7 will survive.

```
set_input_delay 5 -max -rise -add_delay [get_clocks CLK2] [get_ports PAD1]   # will be
overridden
set_input_delay 3 -min -fall -add_delay [get_clocks CLK2] [get_ports PAD1]
set_input_delay 3 -max -fall -add_delay [get_clocks CLK2] [get_ports PAD1]
set_input_delay 7 -max -rise -add_delay [get_clocks CLK2] [get_ports PAD1]
```

See Also

set_output_delay

Tcl Command Documentation Conventions

set_max_delay

Tcl command; specifies the maximum delay for the timing paths in the current scenario.

```
set_max_delay delay_value [-from from_list] [-to to_list] [-through through_list]
```

Arguments

delay_value

Specifies a floating point number in nanoseconds that represents the required maximum delay value for specified paths.

- If the path starting point is on a sequential device, the tool includes clock skew in the computed delay.
- If the path starting point has an input delay specified, the tool adds that delay value to the path delay.
- If the path ending point is on a sequential device, the tool includes clock skew and library setup time in the computed delay.
- If the ending point has an output delay specified, the tool adds that delay to the path delay.

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

-through *through_list*

Specifies a list of pins, ports, cells, or nets through which the timing paths must pass.

Description

This command specifies the required maximum delay for timing paths in the current design. The path length for any startpoint in from_list to any endpoint in to_list must be less than delay_value.

The timing engine automatically derives the individual maximum delay targets from clock waveforms and port input or output delays.

The maximum delay constraint is a timing exception. This constraint overrides the default single cycle timing relationship for one or more timing paths. This constraint also overrides a multicycle path constraint.

You must specify at least one of the -from, -to, or -through arguments for this constraint to be valid.

Examples

The following example sets a maximum delay by constraining all paths from ff1a:CLK or ff1b:CLK to ff2e:D with a delay less than 5 ns:

```
set_max_delay 5 -from {ff1a:CLK ff1b:CLK} -to {ff2e:D}
```

The following example sets a maximum delay by constraining all paths to output ports whose names start by "out" with a delay less than 3.8 ns:

```
set_max_delay 3.8 -to [get_ports out*]
```

See Also

set_min_delay

remove_max_delay

Tcl Command Documentation Conventions

set_min_delay

Tcl command; specifies the minimum delay for the timing paths in the current scenario.

```
set_min_delay delay_value [-from from_list] [-to to_list] [-through through_list]
```

Arguments

delay_value

Specifies a floating point number in nanoseconds that represents the required minimum delay value for specified paths.

- If the path starting point is on a sequential device, the tool includes clock skew in the computed delay.
- If the path starting point has an input delay specified, the tool adds that delay value to the path delay.
- If the path ending point is on a sequential device, the tool includes clock skew and library setup time in the computed delay.
- If the ending point has an output delay specified, the tool adds that delay to the path delay.

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

-through *through_list*

Specifies a list of pins, ports, cells, or nets through which the timing paths must pass.

Description

This command specifies the required minimum delay for timing paths in the current design. The path length for any startpoint in from_list to any endpoint in to_list must be less than delay_value.

The timing engine automatically derives the individual minimum delay targets from clock waveforms and port input or output delays.

The minimum delay constraint is a timing exception. This constraint overrides the default single cycle timing relationship for one or more timing paths. This constraint also overrides a multicycle path constraint.

You must specify at least one of the -from, -to, or -through arguments for this constraint to be valid.

Examples

The following example sets a minimum delay by constraining all paths from ff1a:CLK or ff1b:CLK to ff2e:D with a delay less than 5 ns:

```
set_min_delay 5 -from {ff1a:CLK ff1b:CLK} -to {ff2e:D}
```

The following example sets a minimum delay by constraining all paths to output ports whose names start by "out" with a delay less than 3.8 ns:

```
set_min_delay 3.8 -to [get_ports out*]
```

See Also

set_max_delay

remove_min_delay

Tcl Command Documentation Conventions

set_multicycle_path

Tcl command; defines a path that takes multiple clock cycles in the current scenario.

```
set_multicycle_path ncycles [-setup] [-hold] [-setup_only] [-from from_list] [-through
through_list] [-to to_list]
```

Arguments

ncycles

Specifies an integer value that represents a number of cycles the data path must have for setup or hold check. The value is relative to the starting point or ending point clock, before data is required at the ending point.

-setup

Optional. Applies the cycle value for the setup check only. This option does not affect the hold check. The default hold check will be applied unless you have specified another set_multicycle_path command for the hold value.

-hold

Optional. Applies the cycle value for the hold check only. This option does not affect the setup check.

Note: If you do not specify "-setup" or "-hold", the cycle value is applied to the setup check and the default hold check is performed (*ncycles* -1).

-setup_only

Optional. Specifies that the path multiplier is applied to setup paths only. The default value for hold check (which is 0) is applied.

-from from_list

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-through through_list

Specifies a list of pins or ports through which the multiple cycle paths must pass.

-to to_list

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Description

Setting multiple cycle paths constraint overrides the single cycle timing relationships between sequential elements by specifying the number of cycles that the data path must have for setup or hold checks. If you change the multiplier, it affects both the setup and hold checks.

False path information always takes precedence over multiple cycle path information. A specific maximum delay constraint overrides a general multiple cycle path constraint.

If you specify more than one object within one -through option, the path passes through any of the objects.

You must specify at least one of the -from, -to, or -through arguments for this constraint to be valid.

Exceptions

Multiple priority management is not supported in Microsemi SoC designs. All multiple cycle path constraints are handled with the same priority.

Examples

The following example sets all paths between reg1 and reg2 to 3 cycles for setup check. Hold check is measured at the previous edge of the clock at reg2.

```
set_multicycle_path 3 -from [get_pins {reg1}] -to [get_pins {reg2}]
```

The following example specifies that four cycles are needed for setup check on all paths starting at the registers in the clock domain ck1. Hold check is further specified with two cycles instead of the three cycles that would have been applied otherwise.

```
set_multicycle_path 4 -setup -from [get_clocks {ck1}]
set_multicycle_path 2 -hold -from [get_clocks {ck1}]
```

The following example specifies that four cycles are needed for setup only check on all paths starting at the registers in the clock domain REF_CLK_0.

```
set_multicycle_path -setup_only 4 -from [ get_clocks { REF_CLK_0 } ]
```

See Also

remove_multicycle_path

Tcl Command Documentation Conventions

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 acceptable Values for max_opcode for PolarFire is shown in the below table. Default is slow_lv_lt.

Value	Description
slow_lv_lt	Use slow_lv_lt conditions for Maximum Delay Analysis
slow_hv_lt	Use slow_hv_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 acceptable Values for min_opcode for PolarFire is shown in the below table. Default is fast_hv_lt.

Value	Description
fast_hv_lt	Use fast_hv_lt conditions for Maximum Delay Analysis
slow_hv_lt	Use slow_hv_lt conditions for Maximum Delay Analysis
slow_lv_lt	Use slow_lv_lt conditions for Maximum Delay Analysis

-interclockdomain_analysis *value*

Enables or disables inter-clock domain analysis. Default is *yes*.

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
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.

xamples

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

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](#)

smartpower_add_pin_in_domain

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_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_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
TRUE: Enable glitch filtering, glitch_threshold must be specified
FALSE: Disable glitch filtering.

smartpower_import_vcd	Type	Values	Description
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 <u>Initialize frequencies and probabilities</u> 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

<u>Tcl documentation conventions</u>

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 (PolarFire)	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

Value	Description
false	Does not include the annotation coverage summary in the report

```
{filename}
```
Specifies the name of the report.

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_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
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

Value	Description
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_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 SmartPower only.
mode	Sets the voltage in the operating conditions as specified in the SmartPower mode-specific operating range. Applies to SmartPower only.

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

Value	Description
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

Value	Description
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

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 {<sNVM_address_for_the_second_initialization_client>} \
 -third_stage_uprom_start_address {<uPROM_address_for_the_third_initialization_stage_client>} \
 -third_stage_spi_start_address {<SPI_address_for_the_third_initialization_stage_client>} \
 -third_stage_snvm_start_address {<sNVM_address_for_the_third_initialization_stage_client>} \
 -third_stage_spi_type {<SPIFLASH_NO_BINDING_PLAINTEXT | SPIFLASH_BINDING_DEFAULT |
SPIFLASH_BINDING_UEK1 | SPIFLASH_BINDING_UEK2>}\
 -third_stage_spi_clock_divider {< 1 | 2 | 4 | 6>} \
 -init_timeout {<int_between_1_and_128_seconds>} \
 -auto_calib_timeout {<Auto_Calibration_timeout_value_in_milliseconds>} \
 -broadcast_RAMs {< 0 | 1 >} \
 -custom_cfg_file {<Initialization_file_for_custom_configuration>}
```

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_uprom_start_address

String parameter for the uPROM start address of the third stage initialization client. It is optional.

Specified as a 32-bit hexadecimal string and must be valid uPROM address aligned on a block boundary.

-third_stage_snvm_start_address

String parameter for the sNVM start address of the third stage initialization client. It is optional.

Specified as a 32-bit hexadecimal string and must be valid sNVM address.

-third_stage_SPI_start_address

String parameter for the SPIFLASH start address of the third stage initialization client. It is optional.

Specified as a 32-bit hexadecimal string and must be valid SPIFLASH address.

-third_stage_spi_type

The value must be one of SPIFLASH_NO_BINDING_PLAINTEXT or SPIFLASH_BINDING_DEFAULT or SPIFLASH_BINDING_UEK1 or SPIFLASH_BINDING_UEK2.

This parameter determines the valid value for parameter 'third_stage_start_address'.

-third_stage_spi_clock_divider

Specifies the clock frequency appropriate for the SPIFLASH memory on board. The value can be 1, 2, 4, or 6. The default value is 1 which is 80 MHz. The other values are 2-40 MHz, 4-20 MHz and 6-13.33 Mhz.

`-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.

`-broadcast_RAMs`

Specifies broadcast instructions to initialize RAM's to zero's. Value can be either 0 or 1. It is optional.

`-custom_cfg_file`

Specifies the initialization file for custom configuration. It is optional

Example

Example to initialize data with sNVM client

```
configure_design_initialization_data
  -second_stage_start_address {0x0000aa00}
  -third_stage_uprom_start_address {0x00000000}
  -third_stage_snvm_start_address {0x0000aa00}
  -third_stage_spi_start_address {0x00000400}
  -third_stage_spi_type {SPIFLASH_NO_BINDING_PLAINTEXT}
  -third_stage_spi_clock_divider {4}
  -init_timeout 85
  -auto_calib_timeout {1400}
  -broadcast_RAMs {0}
```

Example to initialize data with uPROM client

```
configure_design_initialization_data
  -second_stage_start_address {0x00000000}
  -third_stage_uprom_start_address {0xfffffee2}
  -third_stage_snvm_start_address {0x00000000}
  -third_stage_spi_start_address {0x00000400}
  -third_stage_spi_type {SPIFLASH_NO_BINDING_PLAINTEXT}
  -third_stage_spi_clock_divider {4}
  -init_timeout 45
  -auto_calib_timeout {2000}
  -broadcast_RAMs {0}
```

Example to initialize data with SPI-FLASH client

```
configure_design_initialization_data
  -second_stage_start_address {0x00000000}
  -third_stage_uprom_start_address {0x00000000}
  -third_stage_snvm_start_address {0x00000000}
  -third_stage_spi_start_address {0x000AC120}
  -third_stage_spi_type {SPIFLASH_BINDING_UEK2}
  -third_stage_spi_clock_divider {2}
  -init_timeout 20
  -auto_calib_timeout {500}
  -broadcast_RAMs {1}
```

See Also

generate_design_initialization_data

configure_ram

Tcl command; configures the Fabric RAM clients in the Fabric RAMs tab of the Design and Memory Initialization tool. The target storage type for the third stage initialization can be specified for each Fabric RAM client in the cfg file specified here.

Note: You must run Generate Design Initialization Data (generate_design_initialization_data) after configuring the Fabric RAMs (configure_ram) and/or Design Initialization (configure_design_initialization_data).

```
configure_ram \
   -cfg file <path_to_configuration_file.cfg>
```

Arguments

-cfg_file *path_to_configuration_file.cfg*

Specifies the path to the configuration file of the Fabric RAM client. It is mandatory.

Example

```
configure_ram \
 -cfg_file
{../../Downloads/mpf_dg0852_liberosocv12p0_df/Libero_Project/TVS_Demo/designer/TVS_Demo/
RAM.cfg}
```

See Also

generate_design_initialization_data

configure_snvm

Tcl command; configures the sNVM clients in the sNVM tab of the Design and Memory Initialization tool. Can specify user sNVM clients using this command.

Note: You must run Generate Design Initialization Data (generate_design_initialization_data) after configuring sNVM (configure_snvm) and/or Design Initialization (configure_design_initialization_data).

```
configure_snvm \
 -cfg_file  <path_to_configuration_file.cfg>
```

Arguments

-cfg_file *path_to_configuration_file.cfg*

Specifies the path to the configuration file of the sNVM client. It is mandatory.

Examples

```
configure_snvm \
 -cfg_file
{../../Downloads/mpf_dg0852_liberosocv12p0_df/Libero_Project/TVS_Demo/designer/TVS_Demo/
SNVM.cfg
```

See Also

generate_design_initialization_data

configure_spiflash

Tcl command; configures the SPI Flash clients in the SPI Flash tab of the Design and Memory Initialization tool. Can specify user SPI FLASH clients using this command.

Note: You must run Generate Design Initialization Data (generate_design_initialization_data) after configuring SPI Flash(configure_spiflash) and/or Design Initialization (configure_design_initialization_data).

```
configure_spiflash \
 -cfg_file <path_to_configuration_file.cfg>
```

Arguments

-cfg_file *path_to_configuration_file.cfg*

Specifies the path to the configuration file of the SPI FLASH client. It is mandatory.

Examples

```
configure_spiflash \
 -cfg_file
{../../Downloads/mpf_dg0852_liberosocv12p0_df/Libero_Project/TVS_Demo/designer/TVS_Demo/
spiflash.cfg}
```

See Also

generate_design_initialization_data

SPM_OTP

Configures the parameters for SPM_OTP.

```
configure_tool \
[-name SPM_OTP] \
[-params permanently_disable_debugging 0 | 1] \
[-params permanently_disable_dpk 0 | 1] \
[-params permanently_disable_factory_access 0 | 1] \
[-params permanently_disable_prog_interfaces 0 | 1] \
[-params permanently_disable_upk1 0 | 1] \
[-params permanently_disable_upk2 0 | 1] \
[-params permanently_write_protect_fabric 0 | 1]
```

The following tables list the parameter names and values.

configure_tool –name {SPM_OTP} parameter:value pair

Name	Type	Value	Description
permanently_disable_debugging	bool	false | true | 0 | 1	Specifies that the SmartDebug access control and reading temperature and voltage sensor settings is either permanently enabled or disabled. A value of true/1 will permanently disable debugging. The default value is false.
permanently_disable_dpk	bool	false | true | 0 | 1	Specifies that the Debug Pass Key is either permanently enabled or disabled. A value of true/1 will permanently disable FlashLock DPK unlocking. The default value is false.
permanently_disable_factory_access	bool	false | true | 0 | 1	Specifies that the access policy for Microsemi factory test mode is either permanently enabled or disabled. A value of true/1 will permanently disable Microsemi factory test mode. The default value is false.

Name	Type	Value	Description
permanently_disable_prog_interfaces	bool	false | true | 0 | 1	Specifies that the Programming interfaces such as Auto Programming, JTAG, SPI Slave are either permanently enabled or disabled. A value of true/1 will permanently disable all of the programming interfaces. The default value is false.
permanently_disable_upk1	bool	false | true | 0 | 1	Specifies that the User Key UPK1 is either permanently enabled or disabled. A value of true/1 will permanently disable FlashLock UPK1 unlocking. The default value is false.
permanently_disable_upk2	bool	false | true | 0 | 1	Specifies that the User Key UPK2 is either permanently enabled or disabled. A value of true/1 will permanently disable FlashLock UPK2 unlocking. The default value is false.
permanently_write_protect_fabric	bool	false | true | 0 | 1	Specifies that the write protection for fabric is either permanently enabled or disabled. A value of the true/1 will make the fabric one-time programmable. The default value is false.

Examples

The following example specifies that SPM_OTP tool is configured to permanently disable user keys UPK1 and UPK2.

```
configure_tool \
  -name {SPM_OTP} \
  -params {permanently_disable_debugging:false} \
  -params {permanently_disable_dpk:false} \
  -params {permanently_disable_factory_access:false} \
  -params {permanently_disable_prog_interfaces:false} \
  -params {permanently_disable_upk1:true} \
  -params {permanently_disable_upk2:true} \
  -params {permanently_write_protect_fabric:false}
```

The following example specifies that SPM_OTP tool is configured to permanently disable programming interfaces.

```
configure_tool \
  -name {SPM_OTP} \
  -params {permanently_disable_debugging:false} \
  -params {permanently_disable_dpk:false} \
  -params {permanently_disable_factory_access:false} \
  -params {permanently_disable_prog_interfaces:true} \
  -params {permanently_disable_upk1:false} \
  -params {permanently_disable_upk2:false} \
  -params {permanently_write_protect_fabric:false}
```

See Also

remove_permanent_locks

configure_uprom

Tcl command; configures the uPROM clients in the uPROM tab of the Design and Memory Initialization tool. Can specify user uPROM clients using this command.

Note: You must run Generate Design Initialization Data (generate_design_initialization_data) after configuring uPROM (configure_uprom) and/or Design Initialization (configure_design_initialization_data).

```
configure_uprom \
 -cfg_file  <path_to_configuration_file.cfg>
```

Arguments

-cfg_file *path_to_configuration_file.cfg*
Specifies the path to the configuration file of the uPROM client. It is mandatory.

Examples

```
configure_uprom \
 -cfg_file
{../../Downloads/mpf_dg0852_liberosocv12p0_df/Libero_Project/TVS_Demo/designer/TVS_Demo/
UPROM.cfg}
```

See Also

generate_design_initialization_data

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

See the online help for more information.

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

remove_permanent_locks

Removes all the locks configured in SPM_OTP. This command can only be used when at least one lock is disabled using SPM_OTP.

```
remove_permanent_locks
```

Example

```
remove_permanent_locks
```

See Also

SPM_OTP

select_programmer

This Tcl command enables the specified programmer and disables all other connected programmers. This command is useful when multiple programmers are connected.

```
select_programmer -programmer_id {programmer_id} –host_name {host_name} –host_port
{host_port}
```

Arguments

`-programmer_id <programmer_id>`

The programmer to be enabled. See the Select Programmer topic in online help for more information.

`–host_name <host_name>`

The host name or IP address. This argument is required for a remote programmer and optional for a local programmer. For local programmer, if specified it must be "localhost".

`–host_port <host_port>`

This argument is required for a remote programmer and optional for a local programmer. If omitted, the default port is used (currently, the default is 80).

For a local host, both "localhost" and its port should be specified or omitted.

Note: The def variable "LOCAL_PROGRAM_DEBUG_SERVER_PORT" is used to set a different default local host port.

Examples

```
select_programmer -programmer_id {00557}
select_programmer -programmer_id {00557} \
-host_name {localhost} \
-host_port {80}
```

See Also

Select Programmer

See the online help for more information.

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 Tcl 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_file parameter must be specified. See below.

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. Note: This is a decimal value of bytes.

```
-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. Note: this is a decimal value.

```
-program {1}
```

Note: Only `program | 1` is supported in this release.

Examples

The following examples show 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

Mfg Part Number	Memory Capacity	Manufacturer	Sector Size
Note: Microsemi currently supports only the devices listed above.			

Note: This version of the programmer does not support SPI Flash security. Device security options such as "Hardware Write Protect" should be disabled for the External SPI Flash device.

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_programming_interface

This Tcl command sets the programming interface.

```
set_programming_interface -interface {JTAG | SPI_SLAVE}
```

Arguments

```
set_programming_interface -interface {JTAG | SPI_SLAVE}
```
Specify the programming interface as JTAG or SPI_SLAVE. The default is JTAG.

See Also

Programming Connectivity and Interface

See the online help for more information.

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

FlashPro Express Tcl Commands

close_project

Closes the FlashPro or FlashPro Express project.

```
close_project
```

Arguments

None

Exceptions

None

Example

```
close_project
```

configure_flashpro3_prg

Changes FlashPro3 programmer settings.

```
configure_flashpro3_prg [-vpump {ON|OFF}] [-clk_mode {discrete_clk|free_running_clk}] [-
force_freq {ON|OFF}] [-freq {freq}]
```

Arguments

–vpump `{ON|OFF}`

Enables FlashPro programmer to drive VPUMP. Set to ON to drive VPUMP.

–clk_mode `{discrete_clk|free_running_clk}`

Specifies free running or discrete TCK.

–force_freq `{ON|OFF}`

Forces the FlashPro software to use the TCK frequency specified by the software rather than the TCK frequency specified in the programmer file.

–freq `{freq}`

Specifies the TCK frequency in MHz.

Exceptions

None

Example

The following example sets the VPUMP option to ON, TCK to free running, and uses the TCK frequency specified in the programmer file (force_freq is set to OFF):

```
configure_flashpro3_prg -vpump {ON} -clk_mode {free_running_clk} -force_freq {OFF} -freq
{4}
```

The following example sets VPUMP to ON, TCK to discrete, forces the FlashPro software to use the TCK frequency specified in the software (-force_freq is set to ON) at a frequency of 2 MHz.

```
configure_flashpro3_prg -vpump {ON} -clk_mode {discrete_clk} -force_freq {ON} -freq {2}
```

configure_flashpro4_prg

Changes FlashPro4 programmer settings.

```
configure_flashpro4_prg [-vpump {ON|OFF}] [-clk_mode {discrete_clk|free_running_clk}] [-
force_freq {ON|OFF}] [-freq {freq}]
```

Arguments

-vpump {ON|OFF}

Enables FlashPro4 programmer to drive VPUMP. Set to ON to drive VPUMP.

-clk_mode {discrete_clk|free_running_clk}

Specifies free running or discrete TCK.

-force_freq {ON|OFF}

Forces the FlashPro software to use the TCK frequency specified by the software rather than the TCK frequency specified in the programmer file.

-freq {freq}

Specifies the TCK frequency in MHz.

Exceptions

None

Example

The following example sets the VPUMP option to ON and uses a free running TCK at a frequency of 4 MHz (force_freq is set to OFF).

```
configure_flashpro4_prg -vpump {ON} -clk_mode {free_running_clk} -force_freq {OFF} -freq
{4}
```

The following example sets the VPUMP option to ON, uses a discrete TCK and sets force_freq to ON at 2 MHz.

```
configure_flashpro4_prg -vpump {ON} -clk_mode {discrete_clk} -force_freq {ON} -freq {2}
```

configure_flashpro5_prg

Tcl command; changes FlashPro5 programmer settings.

```
configure_flashpro5_prg [-vpump {ON|OFF}] [-clk_mode {free_running_clk}]
[-programming_method {jtag | spi_slave}] [-force_freq {ON|OFF}] [-freq {freq}]
```

Arguments

-vpump {ON|OFF}

Enables FlashPro5 programmer to drive VPUMP. Set to ON to drive VPUMP. Default is ON.

-clk_mode {free_running_clk}

Specifies free running TCK. Default is free_running_clk.

-programming_method {jtag | spi_slave}

Specifies the programming method to use. Default is jtag.

Note: spi_slave works only with SmartFusion2 and IGLOO2.

-force_freq {ON|OFF}

Forces the FlashPro software to use the TCK frequency specified by the software rather than the TCK frequency specified in the programmer file. Default is OFF.

-freq {freq}

Specifies the TCK frequency in MHz. Default is 4.

Exceptions

None

Example

The following example sets the VPUMP option to ON and uses a free running TCK at a frequency of 4 MHz (force_freq is set to OFF).

configure_flashpro5_prg -vpump {ON} -clk_mode {free_running_clk} -force_freq {OFF} -freq {4}

The following example sets the VPUMP option to ON, uses a free running TCK and sets force_freq to ON at 2 MHz.

configure_flashpro5_prg -vpump {ON} -clk_mode {free_running_clk} -force_freq {ON} -freq {2}

configure_flashpro6_prg

Tcl command; changes FlashPro6 programmer settings.

```
configure_flashpro6_prg
[-force_freq {ON|OFF}] [-freq {freq}]
```

Arguments

-force_freq {ON|OFF}

Forces the FlashPro software to use the TCK frequency specified by the software rather than the TCK frequency specified in the programmer file. Default is OFF.

-freq {freq}

Specifies the TCK frequency in MHz. Default is 4.

Exceptions

None

Example

The following example sets TCK at a frequency of 4 MHz and sets force_freq to OFF.

configure_flashpro6_prg -force_freq {OFF} -freq {4}

The following example sets TCK at a frequency of 2 MHz and sets force_freq to ON.

configure_flashpro6_prg -force_freq {ON} -freq {2}

create_job_project

Tcl command; creates a Flashpro Express job using the programming job exported from Libero.

```
create_job_project –job_project_location location –job_file path -overwrite 0|1
```

Arguments

-job_project_location location

Specifies the location for your FlashPro Express job project.

-job_file path

Path to the Libero job file that is used as input to create the Flashpro Express job project.

-overwrite 0|1

Set value to 1 to overwrite your existing job project. .

Exceptions

None

Example

The following example creates a job project named test.job in the \fpexpress directory. It does not overwrite the existing job project.

```
 create_job_project \
-job_project_location {D:\fpexpress} \
-job_file {D:\test\designer\test\export\test.job} –overwrite 0\
```

dump_tcl_support

Unloads the list of supported FlashPro or FlashPro Express Tcl commands.

```
dump_tcl_support -file {file}
```

Arguments

```
-file {file}
```

Exceptions

None

Example

The following example dumps your Tcl commands into the file 'tcldump.tcl'

```
dump_tcl_support -file {tcldump.tcl}
```

open_project

Opens a FlashPro or FlashPro Express project.

```
open_project -project {project}
```

Arguments

```
-project {project}
```
Specifies the location and name of the project you wish to open.

Exceptions

None

Example

Opens the 'FPPrj1.pro' project from the FPProject1 directory

```
open_project -project {./FPProject1/FPPrj1.pro}
```

ping_prg

Pings one or more programmers.

```
ping_prg (-name {name})*
```

Arguments

-name {*name*}

Specifies the programmer to be pinged. Repeat this argument for multiple programmers.

Exceptions

None

Example

The following example pings the programmers 'FP300085' and 'FP30086'.

```
ping_prg -name {FP300085} -name {FP300086}
```

refresh_prg_list

Refreshes the programmer list. This is most often used to have FlashPro or FlashPro Express detect a programmer that you have just connected.

```
refresh_prg_list
```

Arguments

None

Exceptions

None

Example

```
refresh_prg_list
```

remove_prg

Removes the programmer from the programmer list.

```
remove_prg (-name {name})*
```

Arguments

-name {*name*}*

Specifies the programmer to be removed. You can repeat this argument for multiple programmers.

Exceptions

None

Example

The following example removes the programmer '03178' from the programmer list:

```
remove_prg (name {03178})*
```

run_selected_actions

Runs the selected action on the specified programmer and returns the exit code from the action. If no programmer name is specified, the action is run on all connected programmers. Only one exit code is

returned, so return code cannot be used when action is run on more than one programmer. A programming file must be loaded.

```
run_selected_actions [(-name {name})*]
```

Arguments

-name {name}

Optional argument that specifies the programmer name. You can repeat this argument for multiple programmers.

Exceptions

None

Example

The following example runs the selected actionS on the programmers 'FP30085' and 'FP30086'.

```
run_selected_actions -name {FP300085} -name {FP300086}
```

Example using return code:

```
if {[catch {run_selected_actions} return_val]} {puts "Error running Action"} else {puts
"exit code $return_val"}
```

Example returning exit code to the command line (returns exit 99 on script failure, otherwise returns exit code from selected action):

```
if {[catch {run_selected_actions} return_val]}{exit 99} else {exit $return_val}
```

save_log

Saves the log file.

```
save_log -file {file}
```

Arguments

-file {file}

Specifies the log filename.

Exceptions

None

Example

The following example saves the log file with the name 'my_logfile1.log':

```
save_log -file {my_logfile1.log}
```

save_project

Saves the FlashPro or FlashPro Express project.

```
save_project
```

Arguments

None

Exceptions

None

Example

```
save_project
```

scan_chain_prg

In single mode, this command runs scan chain on a programmer.

In chain mode, this command runs scan and check chain on a programmer if devices have been added in the grid.

```
scan_chain_prg [(-name {name})+]
```

Arguments

```
-name {name}
```
Specifies the programmer name.

Exceptions

None

Example

The following example runs scan chain on a single programmer (single mode) named '21428':

```
scan_chain_prg -name {21428}
```

self_test_prg

Runs Self-Test on a programmer.

```
self_test_prg (-name {name})*
```

Arguments

```
-name {name}
```
Specifies the programmer name. You can repeat this argument for multiple programmers.

Exceptions

None

Example

The following examples runs the self test on the programmer '30175':

```
self_test_prg (-name {30175})*
```

set_prg_name

Changes the user name of a programmer.

```
set_prg_name -name {name} -new_name {new_name}
```

Arguments

`-name {`*`name`*`}`

Identifies the old programmer name.

`-new_name {`*`new_name`*`}`

Specifies the new programmer name.

Exceptions

None

Example

The following example changes the name of the programmer 'FP300086' to 'FP3Prg2':

`set_prg_name -name {FP300086} -new_name {FP3Prg2}`

set_programming_action

Selects the action for a device. The device name parameter must be specified only in chain programming mode. A programming file must be loaded. The device must be a Microsemidevice.

```
set_programming_action [-name {name}] -action {action}
```

Arguments

`-name {`*`name`*`}`

Specifies the device name.

`-action {`*`action`*`}`

Specifies the action.

Exceptions

Must be a Microsemi device

Example

The following example sets the programming action in single programming mode:

`set_programming_action -action {PROGRAM}`

And in chain programming mode:

`set_programming_action -name {MyDevice1} -action {ERASE}`

set_programming_file

Sets the programming file for a device. Either the *`file`* or the *`no_file`* flag must be specified. A programming file must be loaded. The device must be a Microsemi device .

```
set_programming_file [-name {name}] [-file {file}] [-no_file { }]
```

Arguments

`-name {`*`name`*`}`

Specifies the device name. This argument must be specified only in chain programming mode.

`-file {`*`file`*`}`

Specifies the programming file.

`-no_file`

Specifies to unload the current programming file.

Exceptions

Must be a Microsemi device.

Examples

In single programming mode:

```
set_programming_file -file {e:/design/pdb/TopA3P250.pdb}
```

In chain programming mode:

```
set_programming_file –name {MyDevice2} –file {e:/design/pdb/TopA3P250.pdb}
set_programming_file –name {MyDevice1} –no_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.
plot_eye	Plots eye and exports eye plots.
xcvr_read_register	Reads SCB registers and their field values.
xcvr_write_register	Writes SCB registers and their field values.
Additional Commands	
event_counter	Runs on signals that are assigned to channel A on the live probe, and displays the total events.
export_smart_debug_data	Exports debug data for the SmartDebug application.
fhb_control	Provides FPGA Hardware Breakpoint (FHB) feature capability for SmartDebug.
frequency_monitor	Calculates the frequency of a signal that is assigned to live probe A.
get_programmer_info	Lists the IDs of all FlashPRO programmers connected to the machine.
uprom_read_memory	Reads uPROM memory block from the device.
Standalone SmartDebug Commands	
construct_chain_automatically	Automatically starts chain construction for the specified programmer.
scan_chain_prg	In single mode, this Tcl command runs scan chain on a programmer. In chain mode, this Tcl command runs scan and check chain on a programmer if devices have been added in the grid.
enable_device	Enables or disables a device in the chain.
set_debug_programmer	Sets the debug programmer.
set_device_name	Sets the device name.

Command	Action
Probe	
set_programming_file	Sets the programming file for a device.
set_programming_action	Selects tehe action for a device.
run_selected_actions	Runs the selected action for a 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
```

construct_chain_automatically

This Tcl command automatically starts chain construction for the specified programmer.

```
construct_chain_automatically -name {programmer_name}
```

Arguments

-name

Specify the device (programmer) name. This argument is mandatory.

Example

For a single programmer:

```
construct_chain_automatically -name {21428}
```

See Also

scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_file

set_programming_action

run_selected_actions

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.
```

enable_device

This Tcl command enables or disables a device in the chain. When the device is disabled, it is bypassed. The device must be a Microsemi device.

```
enable_device -name {device_name} -enable {1 | 0}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-enable

Specify the enable device. This argument is mandatory.

Example

```
enable_device -name {MPF300 (T_ES|TS_ES)} -enable 1
```

See Also

construct_chain_automatically

scan_chain_prg

set_debug_programmer

set_device_name

set_programming_file

set_programming_action

run_selected_actions

event_counter

The event_counter Tcl command runs on signals that are assigned to channel A on the live probe, and displays the total events. It can be run before or after setting the live probe signal to channel A. The user specifies the duration to run the event_counter command.

```
event_counter -run -stop -after duration_in_seconds
```

Arguments

-run

Run event_counter.

-stop

Stop event_counter.

-after *duration_in_seconds*

Duration to stop event_counter. Specified by the user. This argument is required when -stop is specified.

Example

```
set_live_probe -probeA {count_out_c[0]:Counter_8bit_0_count_out[0]:Q} -probeB {}
```

```
        event_counter -run
        event_counter -stop -after 10
```

Output

```
Device ID Code = 2F8071CF
The 'read_id_code' command succeeded.
Live probes have been assigned.
Channel A: count_out_c[0]:Counter_8bit_0_count_out[0]:Q
Channel B: Not specified

The 'set_live_probe' command succeeded.

Event Counter = Activated
The 'event_counter' command succeeded.

Event Counter = Stopped
Total Events = 1603561
The 'event_counter' command succeeded.
The Execute Script command succeeded.
```

export_smart_debug_data

Tcl command; exports debug data for the SmartDebug application.

```
export_smart_debug_data [device_components] [bitstream_components] [-file_name {file} [-
export_dir {dir}] [-force_pf_otp 0 | 1]
```

The command corresponds to the Export SmartDebug Data tool in Libero. The command creates a file with the extension "ddc" that contains data based on selected options. This file is used by SmartDebug (standalone application) to create a new SmartDebug project, or it can be imported into a device in SmartDebug (standalone application).

- If you do not specify any design components, all components available in the design will be included by default except the bitstream components.
- The generate_bitstream parameter is required if you want to generate bitstream file and include it in the exported file.
    - You must specify the bitstream components you want to include in the generated bitstream file or all available components will be included.
    - If you choose to include bitstream, and the design has custom security, the custom security bitstream component must be included.

Arguments

device_components

The following device components can be selected. Specify "1" to include the component, and "0" if you do not want to include the component.

```
-probes <1|0>
-package_pins <1|0>
-memory_blocks <1|0>
-envm_data <1|0>
-security_data <1|0>
-chain <1|0>
-programmer_settings <1|0>
-ios_states <1|0>
```

bitstream_components

The following bitstream components can be selected. Specify "1" to include the component, and "0" if you do not want to include the component.

```
        -generate_bitstream <1|0>
        -bitstream_security <1|0>
        -bitstream_fabric <1|0>
        -bitstream_snvm <1|0>
    -file_name file
```

Name of exported file with extension "ddc".

```
    -export_dir dir
```

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

Example

```
export_smart_debug_data \
-file_name "top" \
-export_dir "./" \
-probes 1 \
-package_pins 0 \
-memory_blocks 1 \
-security_data 1 \
-chain 1 \
-programmer_settings 1 \
-ios_states 1 \
-generate_bitstream 1 \
-bitstream_security 0 \
-bitstream_fabric 1 \
-bitstream_snvm 1
```

The following example shows the command with no parameters:

```
export_smart_debug_data
```

fhb_control

This Tcl command provides FPGA Hardware Breakpoint (FHB) feature capability for SmartDebug.

```
fhb_control
    -halt -clock_domain clkDomName(s)/all
    -run -clock_domain clkDomName(s)
    -step number_of_steps -clock_domain clkDomName(s)
    -reset -clock_domain clkDomName(s)
    -arm_trigger -trigger_signal liveProbePoint -trigger_edge_select rising -delay value -
clock_domain clkDomName(s)
    -disarm_trigger -clock_domain clkDomName(s)/all
    -capture_waveform number_of_steps -vcd_file target_file_name
    -clock_domain_status -clock_domain clkDomName(s)/all
```

Arguments

```
    -halt
```

Specifies to halt the clock.

```
    -clock_domain clkDomName(s)/all
```

Specifies clock domain names to halt. Can be single or multiple clock domains, halted in order specified by user.

```
    -run
```

Specifies to run the clock.

```
    -clock_domain clkDomName(s)
```

Specifies clock domain names to run. Can be single or multiple clock domains, releasing the user clock based on order specified.

```
    -step number_of_steps
```

Specifies to step the clock "number_of_steps" times. Minimum value is 1.

-clock_domain *clkDomName(s)*

Specifies clock domain names to step. Can be single or multiple clock domains.

-reset

Specifies to reset FHB configuration for the specified clock domain.

-clock_domain *clkDomName(s)*

Specifies clock domain names to reset. Can be single or multiple clock domains.

-arm_trigger

Specifies to arm FHB configuration for the specified clock domain.

-trigger_signal *liveProbePoint*

Set the trigger signal to arm the FHBs.

-trigger_edge_select *rising*

Specifies the trigger signal edge to arm the FHBs. FHBs will be armed on rising edge of trigger signal.

-delay *value*

-clock_domain *clkDomName(s)*

Specifies clock domain names to be armed by the trigger signal. Can be single or multiple clock domains.

-disarm_trigger

Specifies to disarm FHB configuration for the specified clock domain.

-clock_domain *clkDomName(s)*

Specifies clock domain names to be reset by the trigger signal. Can be single or multiple clock domains.

-capture_waveform *number_of_steps*

Specifies to capture waveform of all the added signals to active probes in the specified clock domain for number_of_steps.

- vcd_file *target_file_name*

Target file to save the data and see the waveform.

-clock_domain_status *clkDomName(s)/all*

Specifies to read and display status of specified clock domain(s). Can be single or multiple clock domains.

Examples

```
fhb_control –halt –clock_domain {"FCCC_0/GL0_INST " "FCCC_0/GL1_INST" }
fhb_control –run –clock_domain {"FCCC_0/GL0_INST " "FCCC_0/GL1_INST" }
fhb_control –step –clock_domain {"FCCC_0/GL0_INST " "FCCC_0/GL1_INST" }
fhb_control –reset –clock_domain {"FCCC_0/GL0_INST " "FCCC_0/GL1_INST" }
fhb_control –arm_trigger –trigger_signal {q_0_c[14]:count_1_q[14]:Q}
–trigger_edge_select {rising} – delay 0 – clock_domain {"FCCC_0/GL0_INST"}
fhb_control –disarm_trigger –trigger_signal {q_0_c[14]:count_1_q[14]:Q}
–trigger_edge_select {rising} – delay 0 – clock_domain {"FCCC_0/GL0_INST"}
fhb_control –capture_waveform {10} –vcd_file {D:/wvf_location/waveform.vcd}
fhb_control – clock_domain_status – clock_domain { "FCCC_0/GL0_INST" "FCCC_0/GL1_INST"
"FCCC_0/GL2_INST" }
```

frequency_monitor

The frequency_monitor Tcl command calculates the frequency of a signal that is assigned to live probe A.

```
run_frequency_monitor -signal signal_name –time duration
```

Arguments

-signal *signal_name*

Specifies the signal name.

-time *duration*

Specifies the duration to run the command. The value can be 0.1, 1, 5, 8, or 10.

Example

```
run_frequency_monitor -signal {count_out_c[7]:Counter_8bit_0_count_out[7]:Q} -time {5}
```

Output

Device ID Code = 2F8071CF
The 'read_id_code' command succeeded.

Frequency = 0.192716 MHz
The 'run_frequency_monitor' command succeeded.
The Execute Script command succeeded.

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
```

optimize_dfe

This Tcl command supports the Optimize DFE feature in SmartDebug.

```
optimize_dfe -dfe_algorithm <type of dfe algorithm> -lane <lane(s) configured in the design>
```

Arguments

-dfe_algorithm

This command executes Dfe Algorithm with type of dfe algorithm and lanes as input. Algorithm selection has two options:

software_based – executes DfeSs.tcl script

xcvr_based –executes internal Dfe Auto Calibration.

This argument is mandatory.

-lane

List of lane(s) configured in the design.

This argument is mandatory.

Examples

```
optimize_dfe -lane {"Q2_LANE0"} -dfe_algorithm {software_based}
optimize_dfe -lane {"Q2_LANE0"} -dfe_algorithm {xcvr_based}
optimize_dfe -lane {"Q2_LANE0" "Q0_LANE0"} -dfe_algorithm {xcvr_based}
```

pcie_config_space

This Tcl command displays the value of the entered parameter in the SmartDebug log window and return thes `register:field` value to the Tcl.

```
pcie_config_space -pcie_block_name {pcie_block_name} -param_name {param name}
```

Arguments

-pcie_block_name {*pcie_block_name*}
Complete logical hierarchy of the PCIE block whose status is to be read from the device. This parameter is mandatory.

-param_name {*param name*}
Parameter name to read from the device. This parameter is mandatory.

Example

```
pcie_config_space -pcie_block_name {sb_0/CM1_Subsystem/my_pcie_0} -param_name
{neg_max_payload}
```

Output Display in SmartDebug window: `512 bytes`

Return value to the tcl script: `0x2`

pcie_ltssm_status

This Tcl command displays the current LTSSM state from the PLDA core in the SmartDebug log window and returns the `register:field` value to the Tcl.

```
pcie_ltssm_status -pcie_block_name {pcie_block_name}
```

Arguments

-pcie_block_name {*pcie_block_name*}
Complete logical hierarchy of the PCIE block whose status is to be read from the device. This parameter is mandatory.

Example

```
pcie_ltssm_status -pcie_block_name {sb_0/CM1_Subsystem/my_pcie_0}
```

Output Display in SmartDebug window: `Configuration.Linkwidth.start`

Return value to the tcl script: `0x2`

plot_eye

This Tcl command is used to plot eye and export eye plots.

```
plot eye -lane {lane_instance_name} -export_dir {location_path}
```

Arguments

-lane
Specify the lane instance name.

-export_dir
Specify the path to the location where the file is to be exported.

Example

```
plot_eye -lane {Q2_LANE0} - export_dir {E:\designs\G5\SERDES\ export.txt}
```

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}
```

run_selected_actions

This Tcl command is used to run the selected action for a device.

```
run_selected_actions
```

This command takes no arguments.

Example

```
set_programming_action -name {MPF300(T_ES|TS_ES)} -action {DEVICE_INFO}
set_programming_action -name {M2S/M2GL090(T|TS|TV)} -action {ERASE}
```

See Also

[construct_chain_automatically](#)
[scan_chain_prg](#)
[enable_device](#)
[set_debug_programmer](#)
[set_device_name](#)
[set_programming_file](#)
[set_programming_action](#)

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"
```

scan_chain_prg

In single mode, this Tcl command runs scan chain on a programmer. In chain mode, this Tcl command runs scan and check chain on a programmer if devices have been added in the grid.

```
scan_chain_prg -name {programmer_name}
```

Arguments

```
-name
```
Specify the device (programmer) name. This argument is mandatory.

Example

```
scan_chain_prg -name {21428}
```

See Also

[construct_chain_automatically](#)
[enable_device](#)

set_debug_programmer

set_device_name.htm

set_programming_file

set_programming_action

run_selected_actions

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]
```

set_debug_programmer

This Tcl command is used to set the debug programmer.

```
set_debug_programmer -name {programmer_name}
```

Arguments

-name

Specify the programmer. This argument is mandatory.

Example

```
set_debug_programmer -name {S201YQST1V}
```

See Also

construct_chain_automatically
scan_chain_prg
enable_device
set_device_name
set_programming_file
set_programming_action
run_selected_actions

set_programming_action

This Tcl command is used to select the action for a device.

```
set_programming_action [-name {device_name}] -action {procedure_action}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-action

Specify the programming action. This argument is mandatory.

Example

```
set_programming_action -name {MPF300(T_ES|TS_ES)} -action {DEVICE_INFO}
set_programming_action -name {M2S/M2GL090(T|TS|TV)} -action {ERASE}
```

See Also

construct_chain_automatically
scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_file

run_selected_actions

set_programming_file

This Tcl command is used to set the programming file for a device. Either the file or the no_file flag must specified. A programming file must be loaded. The device must be a Microsemi device.

```
set_programming_file -name {device_name} -file {stapl_file_name_with_path}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-file

Specify the *file* path. This argument is mandatory.

Example

```
set_programming_file -name {MPF300(T_ES|TS_ES)} -file
{D:/export/CM1_PCIE_TOP_default_uic_12_200_0_12.stp}
```

See Also

construct_chain_automatically
scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_action

run_selected_actions

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 Guide for Libero or the SmartDebug User Guide for PolarFire 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}
```

xcvr_read_register

This Tcl command reads SCB registers and their field values. Read value is in hex format. This command is used
in SmartDebug Signal Integrity.

```
xcvr_read_register -inst_name <inst_name> -reg_name [<reg_name> | <reg_name:field_name>]
```

Arguments

-inst_name <*inst_name*>

Specify the lane instance name used in the design.

-reg_name <*reg_name*> or <*reg_name:field_name*>

Specify the <*reg_name*> for register name or <*reg_name*>:<*field_name*> for the register's field.

Examples

Reading pcslane's 32-bit register LNTV_R0:

```
xcvr_read_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} -reg_name {LNTV_R0}
```

Output:

```
Register Name: LNTV_R0 value: 0x12
The 'xcvr_write_register' command succeeded.
```

Reading Register LNTV_R0 field LNTV_RX_GEAR (i.e. 0th bit of 32-bit register):

```
xcvr_read_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} -reg_name
{LNTV_R0:LNTV_RX_GEAR}
```

Output:
```
Register Name: LNTV_R0:LNTV_RX_GEAR, Value: 0x0
The 'xcvr_read_register' command succeeded.
```

Exception:

SOFT_RESET Register

The SOFT_RESET register is an SCB read/write register containing information such as block ID and Map IDs. It is also used to provide a pulsed reset to the SCB registers. It is a group-specific register.

The SOFT_RESET register is available with the four groups (`pma_lane`, `pma_cmn`, `pcslane`, and `pcscmn`). To read or write this register or its field value, "group name" must be added before "SOFT_RESET".

> `-reg_name <group name>_<SOFT_RESET>` for register name

or

> `[<group name>_<SOFT_RESET>:field_name]` for register field name

where <*group name*> can be PCS, PCSCMN, PMA, or PMA_CMN.

Examples

Reading all four groups' SOFT_RESET register and its field BLOCKID

Reading the PCS SOFT_RESET register and its field BLOCKID (i.e. 16th to 31st bit):
```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCS_SOFT_RESET}
```
Output:
```
Register Name: PCS_SOFT_RESET, Value: 0x300100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:
```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCS_SOFT_RESET:BLOCKID}
```
Output:
```
Register Name: PCS_SOFT_RESET:BLOCKID, Value: 0x30
The 'xcvr_read_register' command succeeded.
```

Reading PCSCMN's SOFT_RESET register and its field BLOCKID (i.e. 16th to 31st bit):
```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCSCMN_SOFT_RESET}
Register Name: PCSCMN_SOFT_RESET, Value: 0x340100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:
```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCSCMN_SOFT_RESET:BLOCKID}
```
Output:
```
Register Name: PCSCMN_SOFT_RESET:BLOCKID, Value: 0x34
The 'xcvr_read_register' command succeeded.
```

Reading PMA's SOFT_RESET register and its field BLOCKID (i.e. 16th to 31st bit):
```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PMA_SOFT_RESET}
```
Output:
```
Register Name: PMA_SOFT_RESET, Value: 0x1300100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_SOFT_RESET:BLOCKID}
```

Output:

```
Register Name: PMA_SOFT_RESET:BLOCKID, Value: 0x130
The 'xcvr_read_register' command succeeded.
```

Reading PMA_CMN's SOFT_RESET register and it's field BLOCKID (i.e. 16th to 31st bit):

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET}
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET, Value: 0x1340100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET:BLOCKID}
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET:BLOCKID, Value: 0x134
The 'xcvr_read_register' command succeeded.
```

See Also

xcvr_write_register

xcvr_write_register

This Tcl command writes SCB registers and their field values. Write value is in hex format. This command is used in SmartDebug Signal Integrity.

```
xcvr_write_register –inst_name <inst_name> –reg_name [<reg_name> | <reg_name:field_name>] –
value {write_value}
```

Arguments

–inst_name <inst_name>

Specify the lane instance name used in the design.

–reg_name <reg_name> or <reg_name:field_name>

Specify the <reg_name> for register name or <reg_name>:<field_name> for the register's field.

-value <write_value>

Specify the value in hex format.

Examples

Writing pcscmn's 32-bit register GSSCLK_CTRL

```
xcvr_write_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} -reg_name {GSSCLK_CTRL} -
value 0xffffffff
```

Output:

```
Register Name: GSSCLK_CTRL value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing Register GSSCLK_CTRL field MCLK_GSSCLK_2_SEL i.e. 16th to 20th bits (5 bits) of 32-bit register

```
xcvr_write_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} \
 -reg_name {GSSCLK_CTRL:MCLK_GSSCLK_2_SEL} -value 0x6
```

Output:
```
Register Name: GSSCLK_CTRL:MCLK_GSSCLK_2_SEL value: 0x6
The 'xcvr_write_register' command succeeded.
```

Exception:

SOFT_RESET Register

The SOFT_RESET register is an SCB read/write register containing information such as block ID and Map IDs. It is also used to provide a pulsed reset to the SCB registers. It is a group-specific register.

The SOFT_RESET register is available with the four groups (pma_lane, pma_cmn, pcslane, and pcscmn). To read or write this register or its field value, "group name" must be added before "SOFT_RESET".

    -reg_name <*group name*>_<*SOFT_RESET*> for register name

or

    [<*group name*>_<*SOFT_RESET*>:field_name] for register field name

where <*group name*> can be PCS, PCSCMN, PMA, or PMA_CMN

Examples

Writing all four groups' SOFT_RESET register and its field PERIPH

Writing to the PCS SOFT_RESET register (32-bits) and its field PERIPH (i.e. 8th bit):
```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCS_SOFT_RESET} -value 0xffffffff
```
Output:
```
Register Name: PCS_SOFT_RESET value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:
```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCS_SOFT_RESET:PERIPH} -value 0x1
```
Output:
```
Register Name: PCS_SOFT_RESET:PERIPH value: 0x1
The 'xcvr_write_register' command succeeded.
```

Writing to PCSCMN's SOFT_RESET register (32-bits) its field PERIPH (i.e. 8th bit):
```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCSCMN_SOFT_RESET} -value 0xffffffff
```
Output:
```
Register Name: PCSCMN_SOFT_RESET value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:
```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCSCMN_SOFT_RESET:PERIPH} -value 0x1
```
Output:
```
Register Name: PCSCMN_SOFT_RESET:PERIPH value: 0x1
The 'xcvr_write_register' command succeeded.
```

Writing to PMA's SOFT_RESET register its field PERIPH (i.e. 8th bit):

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PMA_SOFT_RESET}
-value 0xffffffff
```

Output:

```
Register Name: PMA_SOFT_RESET value: 0xffffffff

The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_SOFT_RESET:PERIPH} -value 0x1
```

Output:

```
Register Name: PMA_SOFT_RESET:PERIPH value: 0x1

The 'xcvr_write_register' command succeeded.
```

Writing to PMA_CMN's SOFT_RESET register its field PERIPH (i.e. 8th bit):

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET} -value 0xffffffff
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET value: 0xffffffff

The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET:PERIPH} -value 0x1
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET:PERIPH value: 0x1

The 'xcvr_write_register' command succeeded.
```

See Also

xcvr_read_register

Configure JTAG Chain Tcl Commands

These commands take a script that contains JTAG chain configuration-specific Tcl commands and passes them to FlashPro Express for execution.

Note that these commands cannot be executed directly from Libero.

add_actel_device

Adds an Actel device to the chain. Either the *file* or *device* parameter must be specified. Chain programming mode must have been set.

```
add_actel_device [-file {filename}] [-device {device}] -name {name}
```

Arguments

Where:

-file{*filename*}

Specifies a programming filename.

-device{*device*}

Specifies the device family (such as MPF300).

-name{*name*}

Specifies the device user name.

Exceptions

None

Example

```
add_actel_device -file {e:/design/stp/TOP.stp} -name {MyDevice1}
add_actel_device -device {MPF300} -name {MyDevice2}
```

add_non_actel_device

Adds a non-Actel device in the chain. Either the file, or (-tck And -ir) parameters must be specified. The Chain programming mode must have been set.

```
add_non_actel_device [-file {file}] [-ir {ir}] [-tck {tck}] [-name {name}]
```

Arguments

-file {*filename*}

Specifies a BSDL file.

-ir {*ir*}

Specifies the IR length.

-tck {*tck*}

Specifies the maximum TCK frequency (in MHz).

-name {*name*}

Specifies the device user name.

Exceptions

None

Examples

```
add_non_actel_device -file {e:/design/bsdl/DeviceX.bsdl } -name {MyDevice3}
add_non_actel_device -ir 8 - tck 5 -name {MyDevice4}
```

add_non_actel_device_to_database

Imports settings via a BSDL file that adds non-Actel or non-Microsemi devices to the device database so that they are recognized during scan chain and auto-construction operations.

```
add_non_actel_device_to_database [-file {bsdl_filename}]
```

Arguments

-file {bsdl_filename}

Specifies the path to the BSDL file and the BSDL filename add to the database.

Supported Families

All non-Microsemi and non-Actel families

Exceptions

N/A

Examples

The following example uses a BSDL file to add a non-Microsemi (1502AS J44) device to the device database:

```
add_non_actel_device_to_database -file {c:/bsdl/atmel/1502AS_J44.bsd}
```

The following example uses a BSDL file to add a non-Microsemi (80200) device to the device database:

```
add_non_actel_device_to_database -file {c:/bsdl/intel/80200_v1.0.bsd}
```

construct_chain_automatically

Automatically starts chain construction for the specified programmer.

```
construct_chain_automatically[(-name {name})+]
```

Arguments

-name {*name*}

Specifies the programmer(s) name(s).

Exceptions

N/A

Example

Example for one programmer:

```
construct_chain_automatically -name {21428}
```

Example for two programmers:

```
construct_chain_automatically -name {21428} -name {00579}
```

copy_device

Copies a device in the chain to the clipboard. Chain programming mode must be set. See the paste_device command for more information.

```
copy_device (-name {name})*
```

Arguments

-name {name}

Specifies the device name. Repeat this argument to copy multiple devices.

Exceptions

None

Example

The example copies the device 'mydevice1' to the same location with a new name 'mydevice2'.

```
copy_device -name {MyDevice1} -name {MyDevice2}
```

cut_device

Removes one or more devices from the chain. It places the removed device in the clipboard. Chain programming mode must be set to use this command. See the paste_device command for more information.

```
cut_device (-name {name})*
```

Arguments

-name {name}

Specifies the device name. You can repeat this argument for multiple devices.

Exceptions

None

Example

The following example removes the devices 'mydevice1' and 'mydevice2' from the chain.

```
cut_device -name {MyDevice1} -name {MyDevice2}
```

enable_device

Enables or disables a device in the chain (if the device is disabled, it is bypassed). Chain programming mode must be set. The device must be a Microsemi device.

```
enable_device -name {name} -enable {TRUE|FALSE}
```

Arguments

`-name {`*`name`*`}`

Specifies your device name

`-enable {TRUE|FALSE}`

Specifies whether the device is to be enabled or disabled. If you specify multiple devices, this argument applies to all specified devices. (TRUE = enable. FALSE = disable)

Exceptions

None

Example

The following example disables the device 'mydevice1' in the chain.

```
enable_device -name {MyDevice1} -enable {FALSE}
```

paste_device

Pastes the devices that are on the clipboard in the chain, immediately above the *position_name* device, if this parameter is specified. Otherwise it places the devices at the end of the chain. The chain programming mode must be enabled.

```
paste_device [-position_name {position_name}]
```

Arguments

`-position_name {`*`position_name`*`}`

Optional argument that specifies the name of a device in the chain.

Exceptions

None

Examples

The following example pastes the devices on the clipboard immediately above the device 'mydevice3' in the chain.

```
paste_device -position_name {MyDevice3}
```

remove_device

Removes the device from the chain. Chain programming mode must be set.

```
remove_device (-name {name})*
```

Arguments

`-name {`*`name`*`}`

Specifies the device name. You can repeat this argument for multiple devices.

Supported Families

All

Exceptions

None

Example

Remove a device 'M2S050T' from the chain:

```
remove_device (-name {M2S050T})*
```

remove_non_actel_device_from_database

Removes settings for non-Microsemi or non-Actel device from the device database.

```
remove_non_actel_device_from_database [-name {device_name}]
```

Arguments

-name {*device_name*}

Specifies the non-Actel or non-Microsemi device name to be removed from the database. You can repeat this argument for multiple devices.

Supported Families

Non-Microsemi and non-Actel devices

Exceptions

None

Example

The following example removes the F1502AS_J44 device from the database:

```
remove_non_actel_device_from_database -name {F1502AS_J44}
```

The following example removes the SA2_PROCESSOR device from the database:

```
remove_non_actel_device_from_database -name {SA2_PROCESSOR}
```

select_libero_design_device

This command selects the Libero design device for the Programming Connectivity and Interface tool within Libero. This command is needed when the tool cannot automatically resolve the Libero design device when there are two or more identical devices that match the Libero design device in the configured JTAG chain.

Syntax

```
select_libero_design_device -name {device_name}
```

Arguments

-name {*device_name*}

Specifies a user-assigned unique device name in the JTAG chain.

Exceptions

None

Example

```
select_libero_design_device -name {M2S050TS (2)}
select_libero_design_device -name {my_design_device}
```

Note

This Tcl command is typically used in a Tcl command script file that is passed to the Libero run_tool command.

```
run_tool –name {CONFIGURE_CHAIN} -script {<flashPro_cmd>.tcl}
```

set_bsdl_file

Sets a BSDL file to a non-Microsemi device in the chain. Chain programming mode must have been set. The device must be a non-Microsemi device.

```
set_bsdl_file -name {name} -file {file}
```

Arguments

name {name}

Specifies the device name.

-file {file}

Specifies the BSDL file.

Supported Families

Any non-Microsemi device supported by FlashPro Express.

Exceptions

None

Example

The following example sets the BSDL file /design/bsdl/NewBSDL2.bsdl to the device 'MyDevice3':

```
set_bsdl_file –name {MyDevice3} –file {e:/design/bsdl/NewBSDL2.bsdl}
```

set_device_ir

Sets the IR length of a non-Microsemi device in the chain. Chain programming mode must be set. The device must be a non-Microsemi device.

```
set_device_ir -name {name} -ir {ir}
```

Arguments

-name {name}

Specifies the device name.

-ir {ir}

Specifies the IR length.

Supported Families

Any non-Microsemi device supported by FlashPro Express.

Exceptions

None

Example

The following example sets the IR length to '2' for the non-Microsemi device 'MyDevice4':

```
set_device_ir -name {MyDevice4} -ir {2}
```

set_device_name

Changes the user name of a device in the chain. Chain programming mode must be set .

```
set_device_name -name {name} -new_name {new_name}
```

Arguments

-name {name}

Identifies the old device name.

-new_name {new_name}

Specifies the new device name.

Exceptions

None

Example

The following example changes the user name of the device from 'MyDevice4' to 'MyDevice5':

```
set_device_name -name {MyDevice4} -new_name {MyDevice5}
```

set_device_order

Sets the order of the devices in the chain to the order specified. Chain programming mode must have been set. Unspecified devices will be at the end of the chain.

```
set_device_order (-name {name})*
```

Arguments

-name {name}

Specifies the device name. To specify a new order you must repeat this argument and specify each device name in the order desired.

Exceptions

None

Example

The following example sets the device order for 'MyDevice1', 'MyDevice2', 'MyDevice3', and 'MyDevice4'. 'MyDevice2' is unspecified so it moves to the end of the chain.

```
set_device_order -name {MyDevice3} -name {MyDevice1} -name {MyDevice4}
```

the new order is:

```
MyDevice3 MyDevice1 MyDevice4 MyDevice2
```

set_device_tck

Sets the maximum TCK frequency of a non-Microsemi device in the chain. Chain programming mode must be set. The device must be a non-Microsemi device.

```
set_device_tck -name {name} -tck {tck}
```

Arguments

`-name {`*`name`*`}`

Specifies the device name.

`-tck {`*`tck`*`}`

Specifies the maximum TCK frequency (in MHz).

Supported Families

Any non-Microsemi device supported by FlashPro Express.

Exceptions

None

Example

The following example sets the maximum TCK frequency of the non-Microsemi device 'MyDevice4':

```
set_device_tck -name {MyDevice4} -tck {2.25}
```

set_device_type

Changes the family of a Microsemi device in the chain. The device must be a Microsemi device. The device parameter below is now optional.

```
set_device_type -name {name} -type {type}
```

Arguments

`-name {`*`name`*`}`

Identifies the name of the device you want to change.

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

Specifies the device family.

Supported Families

Any Microsemi device supported by FlashPro Express.

Exceptions

None

Example

The following example sets the device 'MyDevice2' to the type MPF300.

```
set_device_type -name {MyDevice2} -type {MPF300}
```

set_programming_action

This Tcl command is used to select the action for a device.

```
set_programming_action [-name {device_name}] -action {procedure_action}
```

Arguments

`-name`

Specify the device name. This argument is mandatory.

`-action`

Specify the programming action. This argument is mandatory.

Example

```
set_programming_action -name {MPF300(T_ES|TS_ES)} -action {DEVICE_INFO}
set_programming_action -name {M2S/M2GL090(T|TS|TV)} -action {ERASE}
```

See Also

construct_chain_automatically
scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_file

run_selected_actions

set_programming_file

This Tcl command is used to set the programming file for a device. Either the file or the no_file flag must specified. A programming file must be loaded. The device must be a Microsemi device.

```
set_programming_file -name {device_name} -file {stapl_file_name_with_path}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-file

Specify the *file* path. This argument is mandatory.

Example

```
set_programming_file -name {MPF300(T_ES|TS_ES)} -file
{D:/export/CM1_PCIE_TOP_default_uic_12_200_0_12.stp}
```

See Also

construct_chain_automatically
scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_action

run_selected_actions