Software Debugger

Reference Guide

UG1725 (v2025.2) November 20, 2025

AMD

Table of Contents

Software Debugger (XSDB)

The AMD Vitis™ IDE is a graphical development environment that is helpful when developing for a new processor architecture. It simplifies common functions through logical wizards, so even beginners can use it. However, it is necessary for these tools to be scriptable, meaning they can be modified or extended, providing flexibility. This is especially helpful for developing regression tests that are run daily or when using a specific set of commands frequently. It is particularly useful when developing regression tests that are be run nightly or when running a set of commands that are frequently used.

Software debugger (XSDB) is an interactive and scriptable command-line interface to the Vitis IDE. As with other AMD tools, the scripting language for XSDB is based on the tools command language (Tcl). You can run XSDB commands interactively or script the commands for automation.

XSDB supports debugging and programming, such as:

- Downloading and running applications on hardware targets
- Reading and writing registers
- Setting break points and watch expressions

This reference guide is intended to provide the information you need to develop scripts for software development and debug targeting AMD processors.

In this guide, abbreviations are used for various products produced by AMD. For example:

- Use of `ps7` in the source code implies that these files are targeting the AMD Zynq™ 7000 SoC family of products, and specifically the dual-core Cortex® Arm® A9 processors in the SoC.
- Use of `psu` in the source code implies that this code is targeting an AMD Zynq™ UltraScale+™ MPSoC device, which contains a Cortex quad-core Arm A53, dual-core Arm R5, Arm Mali 400 GPU, and a MicroBlaze™ processor based platform management unit (PMU).
- Hardware definition files (XSA) are used to transfer the information about the hardware system that includes a processor to the embedded software development tools such as Vitis IDE and XSDB. It includes information about which peripherals are instantiated, clocks, memory interfaces, and memory maps.

XSDB Commands

The XSDB commands are described in the following sections.

Target Connection Management

The following is a list of connections commands:

- connect
- disconnect
- targets
- gdbremote connect
- gdbremote disconnect

connect

Connect to hw_server/TCF agent.

Syntax

```
connect [options]
```

Allows users to connect to a server, list connections, or switch between connections.

Options

Option	Description
`-host <host name/ip>`	Name/IP address of the host machine.
`-port <port num>`	TCP port number.
`-url <url>`	URL description of hw_server/TCF agent.
`-list`	List open connections.
`-set <channel-id>`	Set active connection.
`-new`	Create a new connection, even one existing to the same URL.
`-xvc-url <url>`	Open Xilinx virtual cable connection.

Option	Description
`-symbols`	Launch symbol server to enable source-level debugging for remote connections.

Returns

The return value depends on the options used.

`-port, -host, -url, -new`: `<channel-id>` of the new connection or error if the connection fails.

`-list`: List of open channels or nothing when there are no open channels.

`-set`: Nothing.

Examples

```
connect -host localhost -port 3121
```

Connect to hw_server/TCF agent on host localhost and port 3121.

```
connect -url tcp:localhost:3121
```

Identical to the previous example.

disconnect

Disconnect from hw_server/TCF agent.

Syntax

```
disconnect
```

Disconnect from active channel.

```
disconnect <channel-id>
```

Disconnect from specified channel.

Returns

Nothing, if the connection is closed. Error string, if invalid channel-id is specified.

targets

List targets or switch between targets.

Syntax

```
targets [options]
```

List available targets.

```
targets <target id>
```

Select `<target id>` as active target.

Options

Option	Description
`-set`	Set current target to entry single entry in list. This is useful in combination with the -filter option. An error is generated if the list is empty or contains more than one entry.
`-regexp`	Use regexp for filter matching
`-nocase`	Use case insensitive filter matching
`-filter <filter-expression>`	Specify filter expression to control which targets are included in list based on its properties. Filter expressions are similar to Tcl expr syntax. Target properties are referenced by name, while Tcl variables are accessed using the $ syntax string must be quoted. Operators ==, !=, `<=`, >=, `<`, `>`, &&, ||, () are supported. These operators behave like Tcl expr operators. String matching operators =~ and !~ match the LHS string with the RHS pattern using either regexp or string match.
`-target-properties`	Returns a Tcl list of dicts containing target properties.
`-index <index>`	Include targets based on the JTAG scan chain position. This is identical to specifying -filter {jtag_device_index==`<index>`}.
`-timeout <sec>`	Poll until the targets specified by filter option are found on the scan chain, or until timeout. This option is valid only with filter option. This option is useful in case of soft processors on PL, as their initialization and detection takes some time. The timeout value is in seconds. Default timeout is three seconds.

Returns

The return value depends on the options used.

`<none>`: Targets list when no options are used.

`-filter`: Filtered targets list.

`-target-properties`: Tcl list consisting of target properties.

An error is returned when target selection fails.

Examples

```
targets
```

List all targets.

```
targets -filter {name =~ "ARM*#1"}
```

List targets with name starting with "ARM" and ending with "#1".

```
targets 2
```

Set target with id 2 as the current target.

```
targets -set -filter {name =~ "ARM*#1"}
```

Set current target to target with name starting with "ARM" and ending with "#1".

```
targets -set -filter {name =~ "MicroBlaze*"} -index 0
```

Set current target to target with name starting with "MicroBlaze" and which is on the first JTAG device.

gdbremote connect

Connect to GDB remote server.

Syntax

```
gdbremote connect [options] server
```

Connect to a GDB remote server (for example, qemu). xrt_server is used to connect to the remote GDB server.

Options

Option	Description
`-architecture <name>`	Specify default architecture if remote server does not provide it.

Returns

Nothing, if the connection is successful. Error string, if the connection fails.

gdbremote disconnect

Disconnect from GDB remote server.

Syntax

```
gdbremote disconnect [target-id]
```

Disconnect from GDB remote server (for example, qemu).

Returns

Nothing, if the connection is close. Error string, if there is no active connection.

Target Registers

The following is a list of registers commands:

- rrd
- rwr

rrd

Read register for active target.

Syntax

```
rrd [options] [reg]
```

Read registers or register definitions. For a processor core target, the processor core register can be read. For a target representing a group of processor cores, system registers or IOU registers can be read.

Options

Option	Description
`-defs`	Read register definitions instead of values.
`-no-bits`	Does not show bit fields along with register values. By default, bit fields are shown, when available.

Returns

Register names and values, or register definitions if successful. Error string, if the registers cannot be read or if an invalid register is specified.

Examples

```
rrd
```

Read top level registers or groups.

```
rrd r0
```

Read register r0.

```
rrd usr r8
```

Read register r8 in group usr.

rwr

Write to register.

Syntax

```
rwr <reg> <value>
```

Write the `<value>` to active target register specified by `<reg>`. For a processor core target, the processor core register can be written to. For a target representing a group of processor cores, system registers or IOU registers can be written to.

Returns

Nothing, if successful. Error string, if an invalid register is specified or the register cannot be written to.

Examples

```
rwr r8 0x0
```

Write 0x0 to register r8.

```
rwr usr r8 0x0
```

Write 0x0 to register r8 in group usr.

Program Execution

The following is a list of running commands:

- state
- stop
- con

- stp
- nxt
- stpi
- nxti
- stpout
- dis
- print
- locals
- backtrace
- bt
- profile
- mbprofile
- mbtrace

state

Display the current state of the target.

Syntax

```
state
```

Return the current execution state of target.

stop

Stop active target.

Syntax

```
stop
```

Suspend execution of active target.

Returns

Nothing, if the target is suspended. Error string, if the target is already stopped or cannot be stopped.

An information message is printed on the console when the target is suspended.

con

Resume active target.

Syntax

```
con [options]
```

Resume execution of active target.

Options

Option	Description
`-addr <address>`	Resume execution from address specified by `<address>`.
`-block`	Block until the target stops or a timeout is reached.
`-timeout <sec>`	Timeout value in seconds.

Returns

Nothing, if the target is resumed. Error string, if the target is already running or cannot be resumed or does not halt within timeout after being resumed.

An information message is printed on the console when the target is resumed.

Examples

```
con -addr 0x100000
```

Resume execution of the active target from address 0x100000.

```
con -block
```

Resume execution of the active target and wait until the target stops.

```
con -block -timeout 5
```

Resume execution of the active target and wait until the target stops or until the five second timeout is reached.

stp

Step into a line of source code.

Syntax

```
stp [count]
```

Resume execution of the active target until control reaches instruction that belongs to different line of source code. If a function is called, stop at first line of the function code. Error is returned if line number information not available. If <count> is greater than 1, repeat <count> times. Default value of count is 1.

Returns

Nothing, if the target has single stepped. Error string, if the target is already running or cannot be resumed.

An information message is printed on the console when the target stops at the next address.

nxt

Step over a line of source code.

Syntax

```
nxt [count]
```

Resume execution of the active target until control reaches instruction that belongs to a different line of source code, but runs any functions called at full speed. Error is returned if line number information not available. If <count> is greater than 1, repeat <count> times. Default value of count is 1.

Returns

Nothing, if the target has stepped to the next source line. Error string, if the target is already running or cannot be resumed.

An information message is printed on the console when the target stops at the next address.

stpi

Execute a machine instruction.

Syntax

```
stpi [count]
```

Execute a single machine instruction. If the instruction is a function call, stop at the first instruction of the function code. If <count> is greater than 1, repeat <count> times. Default value of count is 1.

Returns

Nothing, if the target has single stepped. Error if the target is already running or cannot be resumed.

An information message is printed on the console when the target stops at the next address.

nxti

Step over a machine instruction.

Syntax

```
nxti [count]
```

Step over a single machine instruction. If the instruction is a function call, execution continues until control returns from the function. If `<count>` is greater than 1, repeat `<count>` times. Default value of count is 1.

Returns

Nothing, if the target has stepped to the next address. Error string, if the target is already running or cannot be resumed.

An information message is printed on the console when the target stops at the next address.

stpout

Step out from current function.

Syntax

```
stpout [count]
```

Resume execution of current target until control returns from current function. If `<count>` is greater than 1, repeat `<count>` times. Default value of count is 1.

Returns

Nothing, if the target has stepped out of the current function. Error if the target is already running or cannot be resumed.

An information message is printed on the console when the target stops at the next address.

dis

Disassemble instructions.

Syntax

```
dis <address> [num]
```

Disassemble `<num>` instructions at address specified by `<address>` The keyword "pc" can be used to disassemble instructions at the current PC. Default value for `<num>` is 1.

Returns

Disassembled instructions if successful. Error string, if the target instructions cannot be read.

Examples

```
dis
```

Disassemble an instruction at the current PC value.

```
dis pc 2
```

Disassemble two instructions at the current PC value.

```
dis 0x0 2
```

Disassemble two instructions at address 0x0.

print

Get or set the value of an expression.

Syntax

```
print [options] [expression]
```

Get or set the value of an expression specified by `<expression>`. The `<expression>` can include constants, local/global variables, CPU registers, or any operator, but pre-processor macros defined through #define are not supported. CPU registers can be specified in the format {$r1}, where r1 is the register name. Elements of complex data types, like structures, can be accessed through the '.' operator. For example, the var1.int_type refers to the int_type element in the var1 struct. Array elements can be accessed through their indices. For example, array1[0] refers to the element at index 0 in array1.

Options

Option	Description
`-add <expression>`	Add the `<expression>` to the auto expression list. The values or definitions of the expressions in the auto expression list are displayed when the expression name is not specified. Frequently used expressions should be added to the auto expression list.
`-defs [expression]`	Return the expression definitions like address, type, size, and RW flags. Not all definitions are available for all the expressions. For example, the address is available only for variables and not when the expression includes an operator.
`-dict [expression]`	Return the result in Tcl dict format, with variable names as dict keys and variable values as dict values. For complex data like structures, names are in the form of parent.child.
`-remove [expression]`	Remove the expression from auto expression list. Only expressions previously added to the list through -add option can be removed. When the expression name is not specified, all the expressions in the auto expression list are removed.
`-set <expression>`	Set the value of a variable. It is not possible to set the value of an expression which includes constants or operators.

Returns

The return value depends on the options used.

`-add or <none>`: Expression value(s)

`-defs`: Expression definition(s)

`-remove or -set`: Nothing.

Error string, if the expression value cannot be read or set.

Examples

```
print Int_Glob
```

Return the value of variable Int_Glob.

```
print -a Microseconds
```

Add the variable Microseconds to auto expression list and return its value.

```
print -a Int_Glob*2 + 1
```

Add the expression (Int_Glob*2 + 1) to auto expression list and return its value.

```
print tmp_var.var1.int_type
```

Return the value of int_type element in var1 struct, where var1 is a member of tmp_var struct.

```
print tmp_var.var1.array1[0]
```

Return the value of the element at index 0 in array array1. array1 is a member of var1 struct, which is in turn a member of tmp_var struct.

```
print
```

Return the values of all the expressions in auto expression list.

```
print -defs
```

Return the definitions of all the expressions in auto expression list.

```
print -set Int_Glob 23
```

Set the value of the variable Int_Glob to 23.

```
print -remove Microseconds
```

Remove the expression Microseconds from auto expression list.

```
print {$r1}
```

Return the value of CPU register r1.

locals

Get or set the value of a local variable.

Syntax

```
locals [options] [variable-name [variable-value]]
```

Get or set the value of a variable specified by `<variable-name>`. When the variable name and value are not specified, values of all the local variables are returned. Elements of complex data types like structures can be accessed through the '.' operator. For example, the var1.int_type refers to the int_type element in the var1 struct. Array elements can be accessed through their indices. For example, array1[0] refers to the element at index 0 in array1.

Options

Option	Description
`-defs`	Return the variable definitions like address, type, size, and RW flags.

Option	Description
`-dict`	Return the result in Tcl dict format, with variable names as dict keys and variable values as dict values. For complex data like structures, names are in the form of parent.child.

Returns

The return value depends on the options used.

`<none>`: Variable value(s)

`-defs`: Variable definition(s)

Nothing, when variable value is set. Error string, if variable value cannot be read or set.

Examples

```
locals Int_Loc
```

Return the value of the local variable Int_Loc.

```
locals
```

Return the values of all the local variables in the current stack frame.

```
locals -defs
```

Return definitions of all the local variables in the current stack frame.

```
locals Int_Loc 23
```

Set the value of the local variable Int_Loc to 23.

```
locals tmp_var.var1.int_type
```

Return the value of the int_type element in the var1 struct, where var1 is a member of the tmp_var struct.

```
locals tmp_var.var1.array1[0]
```

Return the value of the element at index 0 in array array1. array1 is a member of the var1 struct, which is in turn a member of the tmp_var struct.

backtrace

Stack back trace.

Syntax

```
backtrace [options]
```

Return stack trace for current target. Target must be stopped. Use debug information for best result. The alias for backtrace is 'bt' and can be used interchangeably.

Options

Option	Description
`-maxframes <num>`	Maximum number of frames in stack trace. The default value is 10. The actual number of frames could be less depending on program state. To read all the available frames, use -1.

Returns

Stack trace, if successful. Error string, if stack trace cannot be read from the target.

Examples

```
bt
```

Return top 10 frames from stack trace.

```
bt -maxframes 5
```

Return top 5 frames from stack trace.

```
bt -maxframes -1
```

Return all the available frames from stack trace.

bt

Stack back trace.

Syntax

backtrace [options] Return stack trace for current target. Target must be stopped. Use debug information for best result. The alias for backtrace is 'bt' and can be used interchangeably.

Options

Option	Description
`-maxframes <num>`	Maximum number of frames in stack trace. The default value is 10. The actual number of frames could be less depending on program state. To read all the available frames, use -1.

Returns

Stack trace, if successful. Error string, if stack trace cannot be read from the target.

Examples

```
bt
```

Return top 10 frames from stack trace.

```
bt -maxframes 5
```

Return top 5 frames from stack trace.

```
bt -maxframes -1
```

Return all the available frames from stack trace.

profile

Configure and run the GNU profiler.

Syntax

```
profile [options]
```

Configure and run the GNU profiler. Profiling must be enabled while building the BSP and application to be profiled.

Options

Option	Description
`-freq <sampling-freq>`	Sampling frequency.
`-scratchaddr <addr>`	Scratch memory for storing the profiling related data. It needs to be assigned carefully, because it should not overlap with the program sections.
`-out <file-name>`	Name of the output file for writing the profiling data. This option also runs the profiler and collects the data. If a file name is not specified, profiling data is written to gmon.out.

Returns

Depends on options used.

`-scratchaddr, -freq`: Returns nothing on successful configuration. Error string, in case of error.

`-out`: Returns nothing, and generates a file. Error string, in case of error.

Examples

```
profile -freq 10000 -scratchaddr 0
```

Configure the profiler with a sampling frequency of 10000 and scratch memory at 0x0.

```
profile -out testgmon.out
```

Output the profile data in testgmon.out.

mbprofile

Configure and run the MB profiler.

Syntax

```
mbprofile [options]
```

Configure and run the MB profiler, a non-intrusive profiler for profiling the application running on MicroBlaze. The output file is generated in gmon.out format. The results can be viewed using the gprof editor. In case of cycle count, an annotated disassembly file is also generated clearly marking the time taken for execution of instructions.

Options

Option	Description
`-low <addr>`	Low address of the profiling address range.
`-high <addr>`	High address of the profiling address range.
`-freq <value>`	MicroBlaze clock frequency in Hz. Default is 100 MHz.
`-count-instr`	Count number of executed instructions. By default, the number of clock cycles of executed instructions are counted.
`-cumulate`	Cumulative profiling. Profiling without clearing the profiling buffers.
`-start`	Enable and start profiling.
`-stop`	Disable/stop profiling.
`-out <filename>`	Output profiling data to file. `<filename>` Name of the output file for writing the profiling data. If the file name is not specified, profiling data is written to gmon.out.

Returns

Depends on options used. -low, -high, -freq, -count-instr, -start, -cumulate Returns nothing on successful configuration. Error string, in case of error.

`-stop`: Returns nothing, and generates a file. Error string, in case of error.

Examples

```
mbprofile -low 0x0 -high 0x3FFF
```

Configure the mb-profiler with address range 0x0 to 0x3FFF for profiling to count the clock cycles of executed instructions.

```
mbprofile -start
```

Enable and start profiling.

```
mbprofile -stop -out testgmon.out
```

Output the profile data in testgmon.out.

```
mbprofile -count-instr
```

Configure the mb-profiler to profile for entire program address range to count the number of instructions executed.

mbtrace

Configure and run MB trace.

Syntax

```
mbtrace [options]
```

Configure and run MB program and event trace for tracing the application running on MB. The output is the disassembly of the executed program.

Options

Option	Description
`-start`	Enable and start trace. After starting trace the execution of the program is captured for later output.
`-stop`	Stop and output trace.
`-con`	Output trace after resuming execution of active target until a breakpoint is hit. At least one breakpoint or watchpoint must be set to use this option. This option is only available with embedded trace.

Option	Description
`-stp`	Output trace after resuming execution of the active target until control reaches instruction that belongs to different line of source code.
`-nxt`	Output trace after resuming execution of the active target until control reaches instruction that belongs to a different line of source code, but runs any functions called at full speed.
`-out <filename>`	Output trace data to a file. `<filename>` Name of the output file for writing the trace data. If not specified, data is output to standard output.
`-level <level>`	Set the trace level to "full", "flow", "event", or "cycles". If not specified, "flow" is used.
`-halt`	Set to halt program execution when the trace buffer is full. If not specified, trace is stopped but program execution continues.
`-save`	Set to enable capture of load and get instruction new data value.
`-low <addr>`	Set low address of the external trace buffer address range. The address range must indicate an unused accessible memory space. Only used with external trace.
`-high <addr>`	Set high address of the external trace buffer address range. The address range must indicate an unused accessible memory space. Only used with external trace.
`-format <format>`	Set external trace data format to "mdm", "ftm", or "tpiu". If format is not specified, "mdm" is used. The "ftm" and "tpiu" formats are output by Zynq 7000 PS. Only used with external trace.

Returns

Depends on options used. -start, -out, -level, -halt -save, -low, -high, -format Returns nothing on successful configuration. Error string, in case of error.

`-stop, -con, -stp, -nxt`: Returns nothing, and outputs trace data to a file or standard output. Error string, in case of error.

Examples

```
mbtrace -start
```

Enable and start trace.

```
mbtrace -start -level full -halt
```

Enable and start trace, configuring to save complete trace instead of only program flow and to halt execution when trace buffer is full.

```
mbtrace -stop
```

Stop trace and output data to standard output.

```
mbtrace -stop -out trace.out
```

Stop trace and output data to trace.out.

```
mbtrace -con -out trace.out
```

Continue execution and output data to trace.out.

Target Memory

The following is a list of memory commands:

- mrd
- mwr
- osa
- memmap

mrd

Memory read.

Syntax

```
mrd [options] <address> [num]
```

Read `<num>` data values from the active target's memory address specified by `<address>`.

Options

Option	Description
`-force`	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked.

Option	Description
`-size <access-size>`	`<access-size>` can be one of the values below: b = Bytes accesses h = Half-word accesses w = Word accesses d = Double-word accesses Default access size is w. Address is aligned to access-size before reading memory, if the '-unaligned-access' option is not used. For targets that do not support double-word access, the debugger uses two word accesses. If the number of data values to be read is more than 1, the debugger selects the appropriate access size. For example, 1. mrd -size b 0x0 4 Debugger accesses one word from the memory, displays four bytes. 2. mrd -size b 0x0 3 Debugger accesses one half-word and one byte from the memory, displays three bytes. 3. mrd 0x0 3 Debugger accesses three words from the memory and displays three words. To read more than 64 bits of data, specify the number of data words along with the address. Data read is in multiples of access size. For example, to read 128 bits of data, run "mrd -size d `<addr>` 2" or "mrd -size w `<addr>` 4."
`-value`	Return a Tcl list of values, instead of displaying the result on the console.
`-bin`	Return data read from the target in binary format.
`-file <file-name>`	Write binary data read from the target to `<file-name>`.
`-address-space <name>`	Access specified memory space instead default memory space of current target. For Arm® DAP targets, address spaces DPR, APR, and AP`<n>` can be used to access DP registers, AP registers, and MEM-AP addresses respectively. For backwards compatibility, the -arm-dap and -arm-ap options can be used as shorthand for "-address-space APR" and "-address-space AP`<n>`" respectively. The APR address range is 0x0 - 0xfffc, where the higher eight bits select an AP and the lower eight bits are the register address for that AP.
`-unaligned-access`	The memory address is not aligned to the access size before performing a read operation. Support for unaligned accesses is target architecture dependent. If this option is not specified, addresses are automatically aligned to access size.

Note(s)

- Select an APU target to access Arm DAP and MEM-AP address space.

Returns

Memory addresses and data in requested format, if successful. Error string, if the target memory cannot be read.

Examples

```
mrd 0x0
```

Read a word at 0x0.

```
mrd 0x0 10
```

Read 10 words at 0x0.

```
mrd -value 0x0 10
```

Read 10 words at 0x0 and return a Tcl list of values.

```
mrd -size b 0x1 3
```

Read three bytes at address 0x1.

```
mrd -size h 0x2 2
```

Read two half-words at address 0x2.

```
mrd -bin -file mem.bin 0 100
```

Read 100 words at address 0x0 and write the binary data to mem.bin.

```
mrd -address-space APR 0x100
```

Read APB-AP CSW on Zynq. The higher eight bits (0x1) select the APB-AP and the lower eight bits (0x0) are the address of CSW.

```
mrd -address-space APR 0x04
```

Read AHB-AP TAR on Zynq. The higher eight bits (0x0) select the AHB-AP and the lower eight bits (0x4) are the address of TAR.

```
mrd -address-space AP1 0x80090088
```

Read address 0x80090088 on DAP APB-AP. AP 1 selects the APB-AP. 0x80090088 on APB-AP corresponds to DBGDSCR of Cortex-A9#0, on Zynq.

```
mrd -address-space AP0 0xe000d000
```

Read address 0xe000d000 on DAP AHB-AP. AP 0 selects the AHB-AP. 0xe000d000 on AHB-AP corresponds to QSPI device on Zynq.

mwr

Memory write.

Syntax

```
mwr [options] <address> <values> [num]
```

Write <num> data values from list of <values> to active target memory address specified by <address>. If <num> is not specified, all the <values> from the list are written sequentially from the address specified by <address>. If <num> is greater than the size of the <values> list, the last word in the list is filled at the remaining address locations.

```
mwr [options] -bin -file <file-name> <address> [num]
```

Read <num> data values from a binary file and write to active target memory address specified by <address>. If <num> is not specified, all the data from the file is written sequentially from the address specified by <address>.

Options

Option	Description
-force	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked.
-bypass-cache-sync	Do not flush/invalidate CPU caches during memory write. Without this option, the debugger flushes/invalidates caches to make sure caches are in sync.
-size <access-size>	<access-size> can be one of the values below: b = Bytes accesses h = Half-word accesses w = Word accesses d = Double-word accesses Default access size is w. Address is aligned to access-size before writing to memory, if the '-unaligned-access' option is not used. If the target does not support double-word access, the debugger uses two word accesses. If number of data values to be written is more than 1, the debugger selects the appropriate access size. For example, 1. mwr -size b 0x0 {0x0 0x13 0x45 0x56} Debugger writes one word to the memory, combining four bytes. 2. mwr -size b 0x0 {0x0 0x13 0x45} Debugger writes one half-word and one byte to the memory, combining the three bytes. 3. mwr 0x0 {0x0 0x13 0x45} Debugger writes three words to the memory. To write more than 64 bits of data, specify the number of data words along with the address. Data written is in multiples of access size. For example, to write 128 bits of data, run "mwr -size d <addr> 2" or "mwr -size w <addr> 4."
-bin	Read binary data from a file and write it to the target address space.
-file <file-name>	File from which binary data is read, to write to the target address space.
-address-space <name>	Access specified memory space instead default memory space of current target. For Arm DAP targets, address spaces DPR, APR, and AP<n> can be used to access DP registers, AP registers, and MEM-AP addresses respectively. For backwards compatibility, the -arm-dap and -arm-ap options can be used as shorthand for "-address-space APR" and "-address-space AP<n>" respectively. The APR address range is 0x0 - 0xfffc, where the higher eight bits select an AP and the lower eight bits are the register address for that AP.
-unaligned-accesses	Memory address is not aligned to access size before performing a write operation. Support for unaligned accesses is target architecture dependent. If this option is not specified, addresses are automatically aligned to access size.

Note(s)

- Select an APU target to access Arm DAP and MEM-AP address space.

Returns

Nothing, if successful. Error string, if the target memory cannot be written.

Examples

```
mwr 0x0 0x1234
```

Write 0x1234 to address 0x0.

```
mwr 0x0 {0x12 0x23 0x34 0x45}
```

Write four words from the list of values to address 0x0.

```
mwr 0x0 {0x12 0x23 0x34 0x45} 10
```

Write four words from the list of values to address 0x0 and fill the last word from the list at the remaining six address locations.

```
mwr -size b 0x1 {0x1 0x2 0x3} 3
```

Write three bytes from the list at address 0x1.

```
mwr -size h 0x2 {0x1234 0x5678} 2
```

Write two half-words from the list at address 0x2.

```
mwr -bin -file mem.bin 0 100
```

Read 100 words from binary file mem.bin and write the data at target address 0x0.

```
mwr -arm-dap 0x100 0x80000042
```

Write 0x80000042 to APB-AP CSW on Zynq. The higher eight bits (0x1) select the APB-AP and the lower eight bits (0x0) are the address of CSW.

```
mwr -arm-dap 0x04 0xf8000120
```

Write 0xf8000120 to AHB-AP TAR on Zynq. The higher eight bits (0x0) select the AHB-AP and the lower eight bits (0x4) are the address of TAR.

```
mwr -arm-ap 1 0x80090088 0x03186003
```

Write 0x03186003 to address 0x80090088 on DAP APB-AP. AP 1 selects the APB-AP. 0x80090088 on APB-AP corresponds to DBGDSCR of Cortex-A9#0, on Zynq.

```
mwr -arm-ap 0 0xe000d000 0x80020001
```

Write 0x80020001 to address 0xe000d000 on DAP AHB-AP. AP 0 selects the AHB-AP. 0xe000d000 on AHB-AP corresponds to the QSPI device on Zynq.

osa

Configure OS awareness for a symbol file.

Syntax

```
osa -file <file-name> [options]
```

Configure OS awareness for the symbol file `<file-name>` specified. If no symbol file is specified and only one symbol file exists in target's memory map, that symbol file is used. If no symbol file is specified and multiple symbol files exist in target's memory map, an error is thrown.

Options

Option	Description
`-disable`	Disable OS awareness for a symbol file. If this option is not specified, OS awareness is enabled.
`-fast-exec`	Enable fast process start. New processes is not tracked for debug and is not visible in the debug targets view.
`-fast-step`	Enable fast stepping. Only the current process is re-synced after stepping. All other processes are not resynced when this flag is turned on.

Note(s)

- The `<fast-exec>` and `<fast-step>` options are not valid with disable option.

Returns

Nothing, if the OSA is configured successfully. Error, if ambiguous options are specified.

Examples

```
osa -file <symbol-file> -fast-step -fast-exec
```

Enable OSA for `<symbol-file>` and turn on fast-exec and fast-step modes.

```
osa -disable -file <symbol-file>
```

Disable OSA for <symbol-file>.

memmap

Modify memory map.

Syntax

```
memmap <options>
```

Add/remove a memory map entry for the active target.

Options

Option	Description
-addr <memory-address>	Address of the memory region that should be added/ removed from the target's memory map.
-alignment <bytes>	Force alignment during memory accesses for a memory region. If alignment is not specified, default alignment is chosen during memory accesses.
-size <memory-size>	Size of the memory region.
-flags <protection-flags>	Protection flags for the memory region. <protection-flags> can be a bitwise OR of the values below: 0x1 = Read access is allowed. 0x2 = Write access is allowed. 0x4 = Instruction fetch access is allowed. Default value of <protection-flags> is 0x3 (Read/Write Access).
-list	List the memory regions added to the active target's memory map.
-clear	Specify whether the memory region should be removed from the target's memory map.
-relocate-section-map <addr>	Relocate the address map of the program sections to <addr>. This option should be used when the code is self-relocating, so that the debugger can find the debug symbol info for the code. <addr> is the relative address, to which all the program sections are relocated.
-osa	Enable OS awareness for the symbol file. Fast process start and fast stepping options are turned off by default. These options can be enabled using the <osa> command. See "help osa" for more details.
-properties <dict>	Specify advanced memory map properties.
-meta-data <dict>	Specify meta-data of advanced memory map properties.

Note(s)

- Only the memory regions previously added through the memmap command can be removed.

Returns

Nothing, while setting the memory map. A list of memory maps when the -list option is used.

Examples

```
memmap -addr 0xfc000000 -size 0x1000 -flags 3
```

Add the memory region 0xfc000000 - 0xfc000fff to the target's memory map. Read/Write accesses are allowed to this region.

```
memmap -addr 0xfc000000 -clear
```

Remove the previously added memory region at 0xfc000000 from the target's memory map.

Target Download FPGA/BINARY

The following is a list of download commands:

- dow
- verify
- fpga

dow

Download ELF and binary file to target.

Syntax

```
dow [options] <file>
```

Download ELF file `<file>` to active target.

```
dow -data <file> <addr>
```

Download binary file `<file>` to active target address specified by `<addr>`.

Options

Option	Description
`-clear`	Clear uninitialized data (bss).
`-skip-tcm-clear`	When the R5 elfs are part of the PDI and use TCM, PLM initializes TCM before loading the elfs. Debugger does the same when the elfs are loaded through debugger, so that TCM banks are initialized properly. Use this option to skip initializing the TCM.
`-keepsym`	Keep previously downloaded ELFs in the list of symbol files. Default behavior is to clear the old symbol files while downloading an ELF.

Option	Description
`-force`	Overwrite access protection. By default, accesses to reserved and invalid address ranges are blocked.
`-bypass-cache-sync`	Do not flush/invalidate CPU caches during ELF download. Without this option, the debugger flushes/invalidates caches to make sure caches are in sync.
`-relocate-section-map <addr>`	Relocate the address map of the program sections to `<addr>`. This option should be used when the code is self-relocating, so that the debugger can find debug symbol information for the code. `<addr>` is the relative address, to which all the program sections are relocated.
`-vaddr`	Use `<vaddr>` from the ELF program headers while downloading the ELF. This option is valid only for ELF files.

Returns

Nothing.

verify

Verify if ELF/binary file is downloaded correctly to target.

Syntax

```
verify [options] <file>
```

Verify if the ELF file specified by `<file>` is downloaded correctly to the active target.

```
verify -data <file> <addr>
```

Verify if the binary file specified by `<file>` is downloaded correctly to the active target address specified by `<addr>`.

Options

Option	Description
`-force`	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked.
`-vaddr`	Use `<vaddr>` from the ELF program headers while verifying the ELF data. This option is valid only for ELF files.

Returns

Nothing, if successful. Error string, if the memory address cannot be accessed or if there is a mismatch.

fpga

Configure FPGA.

Syntax

```
fpga <bitstream-file>
```

Configure FPGA with given bitstream.

```
fpga [options]
```

Configure FPGA with bitstream specified options, or read FPGA state.

Options

Option	Description
`-file <bitstream-file>`	Specify file containing bitstream.
`-partial`	Configure FPGA without first clearing the current configuration. This option should be used while configuring partial bitstreams created before 2014.3 or any partial bitstreams in binary format.
`-no-revision-check`	Disable bitstream versus silicon revision revision compatibility check.
`-skip-compatibility-check`	Disable bitstream versus FPGA compatibility check.
`-state`	Return whether the FPGA is configured.
`-config-status`	Return configuration status.
`-ir-status`	Return IR capture status.
`-boot-status`	Return boot history status.
`-timer-status`	Return watchdog timer status.
`-cor0-status`	Return configuration option 0 status.
`-cor1-status`	Return configuration option 1 status.
`-wbstar-status`	Return warm boot start address status.

Note(s)

- If no target is selected or if the current target is not a supported FPGA, and only one supported FPGA is found in the targets list, this device is configured.

Returns

Depends on options used.

`-file`, `-partial`: Nothing, if FPGA is configured, or an error if the configuration failed.

One of the other options Configuration value.

Target Reset

The following is a list of reset commands:

- rst

rst

Target reset.

Syntax

```
rst [options]
```

Reset the active target.

Options

Option	Description
`-processor`	Reset the active processor target.
`-cores`	Reset the active processor group. This reset type is supported only on Zynq, Zynq UltraScale+ MPSoC, and Versal devices. A processor group is defined as a set of processor cores and on-chip peripherals like OCM.
`-dap`	Reset Arm DAP. This reset type is supported only with targets that represent Arm DAP. Examples of such targets are APU, RPU, PSU, and Versal.
`-system`	Reset the active system. The is the default reset.
`-srst`	Generate system reset for active target. With JTAG, this is done by generating a pulse on the SRST pin on the JTAG cable associated with the active target.
`-por`	Generate power on reset for active target. With JTAG, this is done by generating a pulse on the POR pin on the JTAG cable associated with the active target.
`-ps`	Generate PS only reset on Zynq UltraScale+ MPSoC. This is supported only through MicroBlaze PMU target.
`-stop`	Suspend cores after reset. If this option is not specified, the debugger choses the default action, which is to resume the cores for -system, and suspend the cores for -processor, and -cores. This option is only supported with the -processor, -cores, and -system options.
`-start`	Resume the cores after reset. See the description of the -stop option for more details.
`-endianness <value>`	Set the data endianness to `<value>`. The following values are supported: le - Little endian; be - Big endian. This option is supported with APU, RPU, A9, A53, and A72 targets. If this option is not specified, the current configuration is not changed.

Option	Description
`-code-endianness <value>`	Set the instruction endianness to `<value>`. The following values are supported: le - Little endian; be - Big endian. This option is supported with APU, RPU, A9, A53, and A72 targets. If this option is not specified, the current configuration is not changed.
`-isa <isa-name>`	Set ISA to `<isa-name>`. Supported isa-names are ARM/A32, A64, and Thumb. This option is supported with APU, RPU, A9, A53, and A72 targets. If this option is not specified, the current configuration is not changed.
`-clear-registers`	Clear CPU registers after a reset is triggered. This option is useful while triggering a reset after the device is powered up. Otherwise, debugger can end up reading invalid system addresses based on the register contents. Clearing the registers avoids unpredictable behavior. This option is supported for Arm targets, when used with '-processor' and '-cores'.
`-type <reset type>`	The following reset types are supported: core, cluster, cpu, dap, system, por, pmc-por, pmc-srst, ps-por, ps-srst, pl-por, and pl-srst. pmc-por, pmc-srst, ps-por, ps-srst, pl-por, and pl-srst are supported for Versal devices. Each of these reset types assert and deassert corresponding bits in RST_PS register of CRP module. pmc-por : RST_PS[PMC_POR] pmc-srst : RST_PS[PMC_SRST] ps-por : RST_PS[PS_POR] ps-srst : RST_PS[PS_SRST] pl-por : RST_PS[PL_POR] pl-srst : RST_PS[PL_SRST]

Note(s)

- For Versal devices, the default subsystem is activated through IPI channel5, before triggering the processor reset. This is needed because PLM does not activate the subsystem when PS ELFs are not part of the PDI. If the IPI channel is not enabled in the Vivado design, the subsystem cannot be activated. This causes runtime issues if PM API are used.

Returns

Nothing, if reset if successful. Error string, if reset is unsupported.

IPI commands to Versal PMC

The following is a list of ipi commands:

- plm

plm

PLM logging.

Syntax

```
plm <sub-command> [options]
```

Configure PLM log-level/log-memory, or copy/retrieve PLM log, based on the sub-command specified. The 'copy-debug-log' sub-command allows you to copy the PLM debug log to user memory. The 'set-debug-log' sub-command allows you to configure the memory for the PLM debug log. The 'set-log-level' sub-command allows you to configure the PLM log level. The 'log' command allows you to retrieve the PLM debug log. Type "help" followed by "plm sub-command", or "plm sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command. Refer to the help for the relevant sub-command for details.

Examples

Refer to the help for the relevant sub-command for details.

plm copy-debug-log

Copy PLM debug log.

Syntax

```
plm copy-debug-log <addr>
```

Copy PLM debug log from debug log buffer to user memory specified by `<addr>`.

Returns

Nothing, if successful. Error, otherwise.

Examples

```
plm copy-debug-log 0x0
```

Copy PLM debug log from the default log buffer to address 0x0.

plm set-debug-log

Configure PLM debug log memory.

Syntax

```
plm set-debug-log <addr> <size>
```

Specify the address and size of the memory which should be used for PLM debug log. By default, PMC RAM is used for PLM debug log.

Returns

Nothing, if successful. Error, otherwise.

Examples

```
plm set-debug-log 0x0 0x4000
```

Use the memory 0x0 - 0x3fff for PLM debug log.

plm set-log-level

Configure PLM log level.

Syntax

```
plm set-log-level <level>
```

Configure the PLM log level. This can be less than or equal to the level set during the compile time. The following levels are supported. 0x1 is for unconditional messages (DEBUG_PRINT_ALWAYS). 0x2 is for general debug messages (DEBUG_GENERAL). 0x3 is for more debug information (DEBUG_INFO). 0x4 is for detailed debug information (DEBUG_DETAILED).

Returns

Nothing, if successful. Error, otherwise.

Examples

```
plm set-log-level 0x1
```

Configure the log level to 1.

plm log

Retrieve the PLM log.

Syntax

```
plm log [options]
```

Retrieve the PLM log, and print it on the console, or a channel.

Options

Option	Description
`-handle <handle>`	Specify the file handle to which the data should be redirected. If no file handle is given, data is printed on stdout.
`-log-mem-addr <addr>`	Specify the memory address from which the PLM log should be retrieved. By default, the address and log size are obtained by triggering IPI commands to PLM. If PLM does not respond to IPI commands, default address 0xf2019000 is used. This option can be used to change default address. If either memory address or log size is specified, then the address and size are not retrieved from PLM. If only one of the address or size options is specified, default value is used for the other option. See below for description about log size.
`-log-size <size in bytes>`	Specify the log buffer size. If this option is not specified, the default size of 1024 bytes is used, only when the log memory information cannot be retrieved from PLM.
`-slr <num>`	Specify the slave slr number. If this option is not specified, the default is SLR0 (master plm). Valid slr range is from 0 to 3.

Returns

Nothing, if successful. Error, otherwise.

Examples

```
set fp [open test.log r]
plm log -handle $fp
```

Retrieve PLM debug log and write it to test.log.

```
plm log -slr 2
```

Retrieve PLM debug log from slave slr 2.

Target Breakpoints/Watchpoints

The following is a list of breakpoints commands:

- bpadd

- bpremove
- bpenable
- bpdisable
- bplist
- bpstatus

bpadd

Set a breakpoint/watchpoint.

Syntax

```
bpadd <options>
```

Set a software or hardware breakpoint at address, function or `<file>:<line>`, or set a read/write watchpoint, or set a cross-trigger breakpoint.

Options

Option	Description
`-addr <breakpoint-address>`	Specify the address at which the breakpoint should be set.
`-file <file-name>`	Specify the `<file-name>` in which the breakpoint should be set.
`-line <line-number>`	Specify the `<line-number>` within the file where the breakpoint should be set.
`-type <breakpoint-type>`	Specify the breakpoint type `<breakpoint-type>` can be one of the values below: auto = Auto-breakpoint type is chosen by the hw_server/TCF agent. This is the default type. hw = hardware breakpoint. sw = software breakpoint.
`-mode <breakpoint-mode>`	Specify the access mode that triggers the breakpoint. `<breakpoint-mode>` can be a bitwise OR of the values below: 0x1 is triggered by a read from the breakpoint location. 0x2 is triggered by a write to the breakpoint location. 0x4 is triggered by an instruction execution at the breakpoint location. This is the default for line and address breakpoints. 0x8 is triggered by a data change (not an explicit write) at the breakpoint location.
`-enable <mode>`	Specify initial enablement state of breakpoint. When `<mode>` is 0 the breakpoint is disabled, otherwise the breakpoint is enabled. The default is enabled.
`-ct-input <list> -ct-output <list>`	Specify input and output cross triggers. `<list>` is a list of numbers identifying the cross trigger pin. For Zynq, 0-7 is CTI for core 0, 8-15 is CTI for core 1, 16-23 is CTI ETB and TPIU, and 24-31 is CTI for FTM.
`-skip-on-step <value>`	Specify the trigger behaviour on stepping. This option is only applicable for cross trigger breakpoints and when DBGACK is used as breakpoint input. 0 = trigger every time core is stopped (default). 1 = suppress trigger on stepping over a code breakpoint. 2 = suppress trigger on any kind of stepping.

Option	Description
`-properties <dict>`	Specify advanced breakpoint properties.
`-meta-data <dict>`	Specify metadata of advanced breakpoint properties.
`-target-id <id>`	Specify a target ID for which the breakpoint should be set. A breakpoint can be set for all the targets by specifying the `<id>` as 'all'. If this option is not used, the breakpoint is set for the active target selected through targets command. If there is no active target, the breakpoint is set for all targets.
`-temp`	The breakpoint is removed after it is triggered once.
`-skip-prologue`	For function breakpoints, the function prologue is skipped while planting the breakpoint.

Note(s)

- Breakpoints can be set in XSDB before connecting to hw_server/TCF agent. If there is an active target when a breakpoint is set, the breakpoint is enabled only for that active target. If there is no active target, the breakpoint is enabled for all the targets. The target-id option can be used to set a breakpoint for a specific target, or all targets. An address breakpoint or a file:line breakpoint can also be set without the options -addr, -file, or -line. For address breakpoints, specify the address as an argument, after all other options. For file:line breakpoints, specify the file name and line number in the format `<file>:<line>`, as an argument, after all other options.

Returns

Breakpoint id or an error if invalid target id is specified.

Examples

```
bpadd -addr 0x100000
```

Set a Breakpoint at address 0x100000. Breakpoint type is chosen by hw_server/TCF agent.

```
bpadd -addr &main
```

Set a function Breakpoint at main. Breakpoint type is chosen by hw_server/TCF agent.

```
bpadd -file test.c -line 23 -type hw
```

Set a hardware breakpoint at test.c:23.

```
bpadd -target-id all 0x100
```

Set a breakpoint for all targets, at address 0x100.

```
bpadd -target-id 2 test.c:23
```

Set a breakpoint for target 2, at line 23 in test.c.

```
bpadd -addr &fooVar -type hw -mode 0x3
```

Set a read/write watchpoint on variable fooVar.

```
bpadd -ct-input 0 -ct-output 8
```

Set a cross trigger to stop Zynq core 1 when core 0 stops.

bpremove

Remove breakpoints/watchpoints.

Syntax

```
bpremove <id-list> | -all
```

Remove the breakpoints/watchpoints specified by `<id-list>` or remove all the breakpoints when the -all option is used.

Options

Option	Description
`-all`	Remove all breakpoints.

Returns

Nothing, if the breakpoint is removed successfully. Error string, if the breakpoint specified by `<id>` is not set.

Examples

```
bpremove 0
```

Remove breakpoint 0.

```
bpremove 1 2
```

Remove breakpoints 1 and 2.

```
bpremove -all
```

Remove all breakpoints.

bpenable

Enable breakpoints/watchpoints.

Syntax

```
bpenable <id-list> | -all
```

Enable the breakpoints/watchpoints specified by `<id-list>` or enable all the breakpoints when -all option is used.

Options

Option	Description
`-all`	Enable all breakpoints.

Returns

Nothing, if the breakpoint is enabled successfully. Error string, if the breakpoint specified by `<id>` is not set.

Examples

```
bpenable 0
```

Enable breakpoint 0.

```
bpenable 1 2
```

Enable breakpoints 1 and 2.

```
bpenable -all
```

Enable all breakpoints.

bpdisable

Disable breakpoints/watchpoints.

Syntax

```
bpdisable <id-list> | -all
```

Disable the breakpoints/watchpoints specified by `<id-list>` or disable all the breakpoints when the -all option is used.

Options

Option	Description
`-all`	Disable all breakpoints.

Returns

Nothing, if the breakpoint is disabled successfully. Error string, if the breakpoint specified by `<id>` is not set.

Examples

```
bpdisable 0
```

Disable breakpoint 0.

```
bpdisable 1 2
```

Disable breakpoints 1 and 2.

```
bpdisable -all
```

Disable all breakpoints.

bplist

List breakpoints/watchpoints.

Syntax

```
bplist
```

List all the breakpoints/watchpoints along with brief status for each breakpoint and the target on which it is set.

Returns

List of breakpoints.

bpstatus

Print breakpoint/watchpoint status.

Syntax

```
bpstatus <id>
```

Print the status of a breakpoint/watchpoint specified by `<id>`. Status includes the target information for which the breakpoint is active and also the breakpoint hit count or error message.

Options

None.

Returns

Breakpoint status, if the breakpoint exists. Error string, if the breakpoint specified by `<id>` is not set.

Jtag UART

The following is a list of streams commands:

- jtagterminal
- readjtaguart

jtagterminal

Start/stop JTAG-based hyperterminal.

Syntax

```
jtagterminal [options]
```

Start/stop a JTAG-based hyperterminal to communicate with the Arm DCC or MDM UART interface.

Options

Option	Description
`-start`	Start the JTAG UART terminal. This is the default option.
`-stop`	Stop the JTAG UART terminal.
`-socket`	Return the socket port number instead of starting the terminal. External terminal programs can be used to connect to this port.

Note(s)

- Select a MDM or Arm/MicroBlaze processor target before running this command.

Returns

Socket port number.

readjtaguart

Start/stop reading from JTAG UART.

Syntax

```
readjtaguart [options]
```

Start/stop reading from the Arm® DCC or MDM UART TX interface. The JTAG UART output can be printed on stdout or redirected to a file.

Options

Option	Description
`-start`	Start reading the JTAG UART output.
`-stop`	Stop reading the JTAG UART output.
`-handle <file-handle>`	Specify the file handle to which the data should be redirected. If no file handle is given, data is printed on stdout.

Note(s)

- Select an MDM or Arm processor target before running this command.
- While running a script in non-interactive mode, the output from JTAG UART cannot be written to the log until "readjtaguart -stop" is used.

Returns

Nothing, if successful. Error string, if data cannot be read from the JTAG UART.

Examples

```
readjtaguart
```

Start reading from the JTAG UART and print the output on stdout.

```
set fp [open test.log w]; readjtaguart -start -handle $fp
```

Start reading from the JTAG UART and print the output to test.log.

```
readjtaguart -stop
```

Stop reading from the JTAG UART.

Miscellaneous

The following is a list of miscellaneous commands:

loadhw

Load a Vivado hardware design.

Syntax

```
loadhw [options]
```

Load a Vivado hardware design, and set the memory map for the current target. If the current target is a parent for a group of processors, the memory map is set for all of its child processors. If current target is a processor, the memory map is set for all the child processors of its parent. This command returns the hardware design object.

Options

Option	Description
`-hw`	Hardware design file.
`-list`	Return a list of open designs for the targets.

Option	Description
`-mem-ranges [list {start1 end1} {start2 end2}]`	List of memory ranges from which the memory map should be set. The memory map is not set for the addresses outside these ranges. If this option is not specified, the memory map is set for all the addresses in the hardware design.

Returns

Design object, if the hardware design is loaded and the memory map is set successfully. Error string, if the hardware design cannot be opened.

Examples

```
targets -filter {name =~ "APU"}; loadhw design.xsa
```

Load the hardware design named design.hdf and set the memory map for all the child processors of the APU target.

```
targets -filter {name =~ "xc7z045"}; loadhw design.xsa
```

Load the hardware design named design.hdf and set the memory map for all the child processors for which xc7z045 is the parent.

loadipxact

Load register definitions from ipxact file.

Syntax

```
loadipxact [options] [ipxact-xml]
```

Load memory mapped register definitions from an ipxact-xml file, or clear previously loaded definitions and return to built-in definitions, or return the XML file that is currently loaded.

Options

Option	Description
`-clear`	Clear definitions loaded from the ipxact file and return to built-in definitions.
`-list`	Return the ipxact file that is currently loaded.

Note(s)

- Select a target that supports physical memory accesses to load memory mapped register definitions. For example, APU, RPU, PSU, and Versal targets support physical memory accesses. Processor cores (A9, R5, A53, A72, etc.) support virtual memory accesses.

Returns

Nothing, if the ipxact file is loaded, or previously loaded definitions are cleared successfully. Error string, if load/clear fails. XML file path if -list option is used, and XML file is previously loaded.

Examples

```
loadipxact <xml-file>
```

Load register definitions from `<xml-file>`. This file should be in ipxact format.

```
loadipxact -clear
```

Clear previously loaded register definitions from an XML file, and return to built-in definitions.

```
loadipxact -list
```

Return the XML file that is currently loaded.

unloadhw

Unload a Vivado hardware design.

Syntax

```
unloadhw
```

Close the Vivado hardware design which was opened during the loadhw command, and clear the memory map for the current target. If the current target is a parent for a group of processors, the memory map is cleared for all its child processors. If the current target is a processor, the memory map is cleared for all the child processors of its parent. This command does not clear the memory map explicitly set by users through the memmap command.

Returns

Nothing.

mdm_drwr

Write to MDM debug register.

Syntax

```
mdm_drwr [options] <cmd> <data> <bitlen>
```

Write to MDM debug register. `<cmd>` is an 8-bit MDM command to access a debug register. `<data>` is the register value and `<bitlen>` is the register width.

Options

Option	Description
`-target-id <id>`	Specify a target id representing the MicroBlaze debug module or MicroBlaze instance to access. If this option is not used and '-user' is not specified, the current target is used.
`-user <bscan number>`	Specify user bscan port number.

Returns

Nothing, if successful.

Examples

```
mdm_drwr 8 0x40 8
```

Write to MDM break/reset control register.

mb_drwr

Write to MicroBlaze debug register.

Syntax

```
mb_drwr [options] <cmd> <data> <bitlen>
```

Write to the MicroBlaze debug register available on MDM. `<cmd>` is an 8-bit MDM command to access a debug register. `<data>` is the register value and `<bitlen>` is the register width.

Options

Option	Description
`-target-id <id>`	Specify a target id representing a MicroBlaze instance to access. If this option is not used and -user is not specified, the current target is used.
`-user <bscan number>`	Specify user bscan port number.
`-which <instance>`	Specify MicroBlaze instance number.

Returns

Nothing, if successful.

Examples

```
mb_drwr 1 0x282 10
```

Write to MB control register.

mdm_drrd

Read from MDM debug register.

Syntax

```
mdm_drrd [options] <cmd> <bitlen>
```

Read an MDM debug register. `<cmd>` is an 8-bit MDM command to access a debug register and `<bitlen>` is the register width. Returns hex register value.

Options

Option	Description
`-target-id <id>`	Specify a target id representing the MicroBlaze debug module or MicroBlaze instance to access. If this option is not used and
`-user is not specified, the current target is used.`	
`-user <bscan number>`	Specify user bscan port number.

Returns

Register value, if successful.

Examples

```
mdm_drrd 0 32
```

Read XMDC ID register.

mb_drrd

Read from MicroBlaze Debug Register.

Syntax

```
mb_drrd [options] <cmd> <bitlen>
```

Read a MicroBlaze Debug Register available on MDM. cmd is 8-bit MDM command to access a Debug Register. bitlen is the register width. Returns hex register value.

Options

Option	Description
`-target-id <id>`	Specify a target id representing MicroBlaze instance to access. If this option is not used and -user is not specified, then the current target is used.
`-user <bscan number>`	Specify user bscan port number.
`-which <instance>`	Specify MicroBlaze instance number.

Returns

Register value, if successful.

Examples

```
mb_drrd 3 28
```

Read MB Status Reg.

configparams

List, get, or set configuration parameters.

Syntax

```
configparams <options>
```

List the name and description for available configuration parameters. Configuration parameters can be global or connection specific, therefore the list of available configuration parameters and their value might change depending on the current connection.

```
configparams <options> <name>
```

Get configuration parameter value(s).

```
configparams <options> <name> <value>
```

Set configuration parameter value.

Options

Option	Description
`-all`	Include values for all contexts in result.
`-context [context]`	Specify context of value to get or set. The default context is "", which represents the global default. Not all options support context-specific values.

Option	Description
`-target-id <id>`	Specify target id or value to get or set. This is an alternative to the -context option.

Returns

Depends on the arguments specified.

`<none>`: List of parameters and description of each parameter.

`<parameter name>`: Parameter value or error, if unsupported parameter is specified.

`<parameter name> <parameter value>`: Nothing if the value is set, or error, if the unsupported parameter is specified.

Examples

```
configparams force-mem-accesses 1
```

Disable access protection for the `<dow>`, `<mrd>`, and `<mwr>` commands.

```
configparams vitis-launch-timeout 100
```

Change the Vitis launch timeout to 100 seconds (used for running Vitis batch mode commands).

version

Get Vitis or hw_server version.

Syntax

```
version [options]
```

Get Vitis or hw_server version. When no option is specified, the Vitis build version is returned.

Options

Option	Description
`-server`	Get the hw_server build version for the active connection.

Returns

Vitis or hw_server version, on success. Error string, if server version is requested when there is no connection.

xsdbserver start

Start XSDB command server.

Syntax

```
xsdbserver start [options]
```

Start XSDB command server listener. The XSDB command server allows external processes to connect to XSDB to evaluate commands. The XSDB server reads commands from the connected socket one line at a time. After evaluation, a line is sent back starting with 'okay' or 'error' followed by the result or error as a backslash quoted string.

Options

Option	Description
`-host <addr>`	Limits the network interface on which to listen for incoming connections.
`-port <port>`	Specifies port to listen on. If this option is not specified, or if the port is zero, a dynamically allocated port number is used.

Returns

Server details are displayed on the console if the server is started successfully. Error string, if a server has been already started.

Examples

```
xsdbserver start
```

Start XSDB server listener using dynamically allocated port.

```
xsdbserver start -host localhost -port 2000
```

Start XSDB server listener using port 2000 and only allow incoming connections on this host.

xsdbserver stop

Stop XSDB command server.

Syntax

```
xsdbserver stop
```

Stop XSDB command server listener and disconnect connected client if any.

Returns

Nothing, if the server is closed successfully. Error string, if the server has not been started already.

xsdbserver disconnect

Disconnect active XSDB server connection.

Syntax

```
xsdbserver disconnect
```

Disconnect current XSDB server connection.

Returns

Nothing, if the connection is closed. Error string, if there is no active connection.

xsdbserver version

Return XSDB command server version.

Syntax

```
xsdbserver version
```

Return XSDB command server protocol version.

Returns

Server version if there is an active connection. Error string, if there is no active connection.

JTAG Access

The following is a list of jtag commands:

- jtag targets
- jtag sequence
- jtag device_properties
- jtag lock
- jtag unlock

- jtag claim
- jtag disclaim
- jtag frequency
- jtag skew
- jtag servers

jtag targets

List JTAG targets or switch between JTAG targets.

Syntax

```
jtag targets
```

List available JTAG targets.

```
jtag targets <target id>
```

Select `<target id>` as active JTAG target.

Options

Option	Description
`-set`	Set current target to entry single entry in list. This is useful in combination with -filter option. An error is generated if list is empty or contains more than one entry.
`-regexp`	Use regexp for filter matching.
`-nocase`	Use case insensitive filter matching.
`-filter <filter-expression>`	Specify filter expression to control that targets are included in list based on its properties. Filter expressions are similar to Tcl expr syntax. Target properties are referenced by name, while Tcl variables are accessed using the $ syntax, string must be quoted. Operators ==, !=, `<=`, `>=`, `<`, `>`, &&, ||, and () are supported. There operators behave like Tcl expr operators. String matching operator =~ and !~ match LHS string with RHS pattern using either regexp or string match.
`-target-properties`	Returns a Tcl list of dictionaries containing target properties.
`-open`	Open all targets in list. List can be shorted by specifying target-ids and using filters.
`-close`	Close all targets in list. List can be shorted by specifying target-ids and using filters.
`-timeout <sec>`	Poll until the targets specified by filter option are found on the scan chain, or until timeout. This option is valid only with filter option. The timeout value is in seconds. Default timeout is three seconds.

Returns

The return value depends on the options used.

`<none>`: JTAG targets list when no options are used.

`-filter`: Filtered JTAG targets list.

`-target-properties`: Tcl list consisting of JTAG target properties.

An error is returned when JTAG target selection fails.

Examples

```
jtag targets
```

List all targets.

```
jtag targets -filter {name == "arm_dap"}
```

List targets with name "arm_dap."

```
jtag targets 2
```

Set target with id 2 as the current target.

```
jtag targets -set -filter {name =~ "arm*"}
```

Set current target to target with name starting with "arm."

```
jtag targets -set -filter {level == 0}
```

List JTAG cables.

jtag sequence

Create JTAG sequence object.

Syntax

```
jtag sequence
```

Create JTAG sequence object. DESCRIPTION The jtag sequence command creates a new sequence object. After creation the sequence is empty. The following sequence object commands are available:

```
sequence state new-state [count]
```

Move JTAG state machine to <new-state> and then generate <count> JTAG clocks. If <clock> is given and <new-state> is not a looping state (RESET, IDLE, IRSHIFT, IRPAUSE, DRSHIFT or DRPAUSE), the state machine moves towards RESET state.

```
sequence irshift [options] [bits [data]]
```

sequence drshift [options] bits [data] Shift data in IRSHIFT or DRSHIFT state. Data is either given as the last argument or if -tdi option is given then data is all zeros or all ones depending on the argument given to -tdi. The <bits> and <data> arguments are not used for irshift when the -register option is specified. Available options:

- -register <name> Select instruction register by name. This option is only supported for irshift. -tdi <value> TDI value to use for all clocks in SHIFT state.
- -binary Format of <data> is binary, for example data from a file or from binary format.
- -integer Format of <data> is an integer. The least significant bit of data is shifted first.
- -bits Format of <data> is a binary text string. The first bit in the string is shifted first.
- -hex Format of <data> is a hexadecimal text string. The least significant bit of the first byte in the string is shifted first.
- -capture Capture TDO data during shift and return from sequence run command.
- -state <new-state> State to enter after shift is complete. The default is RESET.

```
sequence delay usec
```

Generate the delay between sequence commands. No JTAG clocks are generated during the delay. The delay is guaranteed to be at least <usec> microseconds, but can be longer for cables that do not support delays without generating JTAG clocks.

```
sequence get_pin pin
```

Get value of <pin>. Supported pins are cable specific.

```
sequence set_pin pin value
```

Set value of <pin> to <value>. Supported pins are cable specific.

```
sequence atomic enable
```

Set or clear atomic sequences. This is useful to creating sequences that are guaranteed to run with precise timing or fail. Atomic sequences should be as short as possible to minimize the risk of failure.

```
sequence run [options]
```

Run JTAG operations in sequence for the currently selected jtag target. This command is return the result from shift commands using -capture option and from get_pin commands.

Available options are listed as follow-

- binary Format return value(s) as binary. The first bit shifted out is the least significant bit in the first byte returned.
- -integer Format return values(s) as integer. The first bit shifted out is the least significant bit of the integer.
- -bits Format return value(s) as binary text string. The first bit shifted out is the first character in the string.
- -hex Format return value(s) as hexadecimal text string. The first bit shifted out is the least significant bit of the first byte of the in the string.
- -single Combine all return values as a single piece of data. Without this option the return value is a list with one entry for every shift with -capture and every get_pin.

```
sequence clear
```

Remove all commands from sequence.

```
sequence delete
```

Delete sequence.

Returns

JTAG sequence object.

Examples

```
set seqname [jtag sequence]
$seqname state RESET
$seqname drshift -capture -tdi 0 256
set result [$seqname run]
$seqname delete
```

jtag device_properties

Get/set device properties.

Syntax

```
jtag device_properties idcode
```

Get JTAG device properties associated with `<idcode>`.

```
jtag device_properties key value ...
```

Set JTAG device properties.

Returns

Jtag device properties for the given idcode, or nothing, if the idcode is unknown.

Examples

```
jtag device_properties 0x4ba00477
```

Return Tcl dict containing device properties for idcode 0x4ba00477.

```
jtag device_properties {idcode 0x4ba00477 mask 0xffffffff name dap irlen 4}
```

Set device properties for idcode 0x4ba00477.

jtag lock

Lock JTAG scan chain.

Syntax

```
jtag lock [timeout]
```

Lock JTAG scan chain containing current JTAG target. DESCRIPTION Wait for scan chain lock to be available and then lock it. If `<timeout>` is specified the wait time is limited to `<timeout>` milliseconds. The JTAG lock prevents other clients from performing any JTAG shifts or state changes on the scan chain. Other scan chains can be used in parallel. The jtag run_sequence command ensures that all commands in the sequence are performed in order so the use of jtag lock is only needed when multiple jtag run_sequence commands needs to be done without interruption.

Note(s)

- A client should avoid locking more than one scan chain as this can cause dead-lock.

Returns

Nothing.

jtag unlock

Unlock JTAG scan chain.

Syntax

```
jtag unlock
```

Unlock JTAG scan chain containing current JTAG target.

Returns

Nothing.

jtag claim

Claim JTAG device.

Syntax

```
jtag claim <mask>
```

Set claim mask for current JTAG device.

DESCRIPTION- This command attempts to set the claim mask for the current JTAG device. If any set bits in `<mask>` are already set in the claim mask then this command returns error-

```
"already claimed".
```

The claim mask allow clients to negotiate control over JTAG devices. This is different from jtag lock in that 1) it is specific to a device in the scan chain, and 2) any clients can perform JTAG operations while the claim is in effect.

Note(s)

- Currently claim is used to disable the hw_server debugger from controlling microprocessors on Arm DAP devices and FPGA devices containing MicroBlaze processors.

Returns

Nothing.

jtag disclaim

Disclaim JTAG device.

Syntax

```
jtag disclaim <mask>
```

Clear claim mask for current JTAG device.

Returns

Nothing.

jtag frequency

Get/set JTAG frequency.

Syntax

```
jtag frequency
```

Get JTAG clock frequency for current scan chain.

```
jtag frequency -list
```

Get list of supported JTAG clock frequencies for current scan chain.

```
jtag frequency <frequency>
```

Set JTAG clock frequency for current scan chain. This frequency is persistent as long as the hw_server is running, and is reset to the default value when a new hw_server is started.

Returns

Current JTAG frequency, if no arguments are specified, or if JTAG frequency is successfully set. Supported JTAG frequencies, if -list option is used. Error string, if invalid frequency is specified or frequency cannot be set.

jtag skew

Get/set JTAG skew.

Syntax

```
jtag skew
```

Get JTAG clock skew for current scan chain.

```
jtag skew <clock-skew>
```

Set JTAG clock skew for current scan chain.

Note(s)

- Clock skew property is not supported by some JTAG cables.

Returns

Current JTAG clock skew, if no arguments are specified, or if JTAG skew is successfully set. Error string, if invalid skew is specified or skew cannot be set.

jtag servers

List, open or close JTAG servers.

Syntax

```
jtag servers [options]
```

List, open, and close JTAG servers. JTAG servers are use to implement support for different types of JTAG cables. An open JTAG server will enumerate or connect to available JTAG ports.

Options

Option	Description
`-list`	List opened servers. This is the default if no other option is given.
`-format`	Return the format of each supported server string.
`-open <server>`	Specifies server to open.
`-close <server>`	Specifies server to close.

Returns

Depends on the options specified.

`<none>`, -list: List of open JTAG servers.

`-format`: List of supported JTAG servers.

`-close`: Nothing if the server is closed, or an error string, if invalid server is specified.

Examples

```
jtag servers
```

List opened servers and number of associated ports.

```
jtag servers -open xilinx-xvc:localhost:10200
```

Connect to XVC server on host localhost port 10200.

```
jtag servers -close xilinx-xvc:localhost:10200
```

Close XVC server for host localhost port 10200.

Target File System

The following is a list of tfile commands:

tfile open

Open file.

Syntax

```
tfile open <path>
```

Open specified file.

Returns

File handle.

tfile close

Close file handle.

Syntax

```
tfile close <handle>
```

Close specified file handle.

Returns

Nothing.

tfile read

Read file handle.

Syntax

```
tfile read <handle>
```

Read from specified file handle.

Options

Option	Description
`-offset <seek>`	File offset to read from.

Returns

Read data.

tfile write

Write file handle.

Syntax

```
tfile write <handle>
```

Write to specified file handle.

Options

Option	Description
`-offset <seek>`	File offset to write to.

Returns

Nothing.

tfile stat

Get file attributes from path.

Syntax

```
tfile stat <handle>
```

Get file attributes for `<path>`.

Returns

File attributes.

tfile lstat

Get link file attributes from path.

Syntax

```
tfile lstat <path>
```

Get link file attributes for `<path>`.

Returns

Link file attributes.

tfile fstat

Get file attributes from handle.

Syntax

```
tfile fstat <handle>
```

Get file attributes for `<handle>`.

Returns

File attributes.

tfile setstat

Set file attributes for path.

Syntax

```
tfile setstat <path> <attributes>
```

Set file attributes for `<path>`.

Returns

File attributes.

tfile fsetstat

Set file attributes for handle.

Syntax

```
tfile fsetstat <handle> <attributes>
```

Set file attributes for `<handle>`.

Returns

File attributes.

tfile remove

Remove path.

Syntax

```
tfile remove <path>
```

Remove `<path>`.

Returns

Nothing.

tfile rmdir

Remove directory.

Syntax

```
tfile rmdir <path>
```

Remove directory `<path>`.

Returns

Nothing.

tfile mkdir

Create directory.

Syntax

```
tfile mkdir <path>
```

Make directory `<path>`.

Returns

Nothing.

tfile realpath

Get real path.

Syntax

```
tfile realpath <path>
```

Get real path of `<path>`.

Returns

Real path.

tfile rename

Rename path.

Syntax

```
tfile rename <old path> <new path>
```

Rename file or directory.

Returns

Nothing.

tfile readlink

Read symbolic link.

Syntax

```
tfile readlink <path>
```

Read link file.

Returns

Target path.

tfile symlink

Create symbolic link.

Syntax

```
tfile symlink <old path> <new path>
```

Symlink file or directory.

Returns

Nothing.

tfile opendir

Open directory.

Syntax

```
tfile opendir <path>
```

Open directory `<path>`.

Returns

File handle.

tfile readdir

Read directory.

Syntax

```
tfile readdir <file handle>
```

Read directory.

Returns

File handle.

tfile copy

Copy target file.

Syntax

```
tfile copy <src> <dest>
```

Copy file `<src>` to `<dest>`.

Returns

Copy file locally on target.

tfile user

Get user attributes.

Syntax

```
tfile user
```

Get user attributes.

Returns

User information.

tfile roots

Get file system roots.

Syntax

```
tfile roots
```

Get file system roots.

Returns

List of file system roots.

tfile ls

List directory contents.

Syntax

```
tfile ls <path>
```

List directory contents.

Returns

Directory contents.

SVF Operations

The following is a list of svf commands:

- svf config
- svf generate

- svf mwr
- svf dow
- svf stop
- svf con
- svf delay
- svf rst

svf config

Configure options for SVF file.

Syntax

```
svf config [options]
```

Configure and generate SVF file.

Options

Option	Description
-scan-chain <list of idcode-irlength pairs>	List of idcode-irlength pairs. This can be obtained from xsdb command - jtag targets
-device-index <index>	This is used to select device in the jtag scan chain.
-cpu-index <processor core>	Specify the cpu-index to generate the SVF file. For A53#0 - A53#3 on ZynqMP, use cpu-index 0 -3 For R5#0 - R5#1 on ZynqMP, use cpu-index 4 -5 For A9#0 - A9#1 on Zynq, use cpu-index 0 -1 If multiple MicroBlaze processors are connected to MDM, select the specific MicroBlaze index for execution.
-out <filename>	Output SVF file.
-delay <tcks>	Delay in ticks between AP writes.
-linkdap	Generate SVF for linking DAP to the jtag chain for ZynqMP Silicon versions 2.0 and above.
-bscan <user port>	This is used to specify user bscan port to which MDM is connected.
-mb-chunksize <size in bytes>	This option is used to specify the chunk size in bytes for each transaction while downloading. Supported only for MicroBlaze processors.
-exec-mode	Execution mode for Arm v8 cores. Supported modes are a32 (v8 core is set up in 32-bit mode) and a64 (v8 core is set up in 64-bit mode).

Returns

Nothing.

Examples

```
svf config -scan-chain {0x14738093 12 0x5ba00477 4} -device-index 1  -cpu-
index 0 -out "test.svf"
```

This creates a SVF file with name test.svf for core A53#0

```
svf config -scan-chain {0x14738093 12 0x5ba00477 4} -device-index 0  -bscan
pmu -cpu-index 0 -out "test.svf"
```

This creates a SVF file with name test.svf for PMU MB

```
svf config -scan-chain {0x23651093 6} -device-index 0 -cpu-index 0  -bscan
user1 -out "test.svf"
```

This creates a SVF file with name test.svf for MB connected to MDM on bscan USER1

svf generate

Generate recorded SVF file.

Syntax

```
svf generate
```

Generate SVF file in the path specified in the config command.

Options

None.

Returns

If successful, this command returns nothing. Otherwise it returns an error.

Examples

```
svf generate
```

svf mwr

Record memory write to SVF file.

Syntax

```
svf mwr <address> <value>
```

Write `<value>` to the memory address specified by `<address>`.

Options

None.

Returns

If successful, this command returns nothing. Otherwise it returns an error.

Examples

```
svf mwr 0xffff0000 0x14000000
```

svf dow

Record elf download to SVF file.

Syntax

```
svf dow <elf file>
```

Record downloading of elf file `<elf file>` to the memory.

```
svf dow -data <file> <addr>
```

Record downloading of binary file `<file>` to the memory.

Options

None.

Returns

If successful, this command returns nothing. Otherwise it returns an error.

Examples

```
svf dow "fsbl.elf"
```

Record downloading of elf file fsbl.elf.

```
svf dow -data "data.bin" 0x1000
```

Record downloading of binary file data.bin to the address 0x1000.

svf stop

Record stopping of core to SVF file.

Syntax

```
svf stop
```

Record suspending execution of current target to SVF file.

Options

None.

Returns

Nothing.

Examples

```
svf stop
```

svf con

Record resuming of core to SVF file.

Syntax

```
svf con
```

Record resuming the execution of active target to SVF file.

Options

None.

Returns

Nothing.

Examples

```
svf con
```

svf delay

Record delay in tcks to SVF file.

Syntax

```
svf delay <delay in tcks>
```

Record delay in tcks to SVF file.

Options

None.

Returns

Nothing.

Examples

```
svf delay 1000
```

Delay of 1000 tcks is added to the SVF file.

svf rst

Reset

Syntax

```
svf rst
```

System Reset

Options

None.

Returns

If successful, this command returns nothing. Otherwise it returns an error.

Examples

```
svf rst
```

Device Configuration System

The following is a list of device commands:

- device program
- device status

- device authjtag

device program

Program PDI/BIT.

Syntax

```
device program <file>
```

Program PDI or BIT file into the device.

Note(s)

- If no target is selected or if the current target is not a configurable device, and only one supported device is found in the targets list, then this device will be configured. Otherwise, users will have to select a device using targets command.
- device program command is currently supported for Versal devices only. Other devices will be supported in future releases.
- For Versal devices, users can run "plm log" to retrieve plm log from memory.

Returns

Nothing, if device is configured, or an error if the configuration failed.

device status

Return JTAG register status.

Syntax

```
device status [options] <jtag-register-name>
```

Return device JTAG Register status, or list of available registers if no name is given.

Options

Option	Description
`-jreg-name <jtag-register-name>`	Specify jtag register name to read. This is the default option, so register name can be directly specified as an argument without using this option.
`-jtag-target <jtag-target-id>`	Specify jtag target id to use instead of the current target. This is primarily used when there isn't a valid target option.
`-hex`	Format the return data in hexadecimal.
`-slr <slr-index>`	Select the SLR from which to read the register (default SLR 0).

Returns

Status report.

device authjtag

Secure debug BIN.

Syntax

```
device authjtag <file>
```

Unlock device for secure debug.

Options

Option	Description
`-jtag-target <jtag-target-id>`	Specify jtag target id to use instead of the current target. This is primarily used when there isn't a valid target option.

Note(s)

- If no target is selected or if the current target is not a configurable device, and only one supported device is found in the targets list, then this device will be configured. Otherwise, users will have to select a device using targets command.
- device authjtag command is currently supported for Versal devices only.

Returns

Nothing, if secure debug is successful, or an error if failed.

STAPL Operations

The following is a list of stapl commands:

- stapl config
- stapl start
- stapl stop

stapl config

Configure stapl target.

Syntax

```
stapl config <options>
```

Create a hw_target (jtag chain) and add all the hw_devices given in the scan-chain list to the hw_target. It also configures the stapl output file where the stapl data is recorded.

Options

Option	Description
`-out <filepath>`	Output file path. Only one of the -out and -handle options should be used. If the -out option is provided, the file will be explicitly opened in a+ mode.
`-handle <filehandle>`	File handle returned by open command for output. Only one of the -out and -handle options should be used.
`-scan-chain <list-of-dicts>`	List of devices in the scan-chain. Each list element must be a dict of device properties in the format {name `<string>` idcode `<int>` irlen `<int>` idcode2 `<int>` mask `<int>`}. For example: [list [dict create name `<device1_name>` idcode `<idcode>` irlen `<irlen>` idcode2 `<idcode2>` mask `<mask>`] [dict create name `<device2_name>` idcode `<idcode>` irlen `<irlen>` idcode2 `<idcode2>` mask `<mask>`]] The order of devices specified with scan-chain option should match the order of devices on the physical hardware where the stapl file is played back. Only one of the -scan-chain and -part options should be used.
`-part <device-name list>`	List of part names of the AMD devices to add to the scan-chain. This option works only with AMD devices. This option can be used instead of the -scan-chain option.
`-checksum`	Calculate stapl-data CRC and append it to the stapl file. If not specified, CRC 0 is appended.

Note(s)

- For AMD devices, if the device_name or idcode is specified in the scan-chain information, the other parameters are optional. All the JTAG TAPs are added automatically to the scan-chain for AMD devices.

Returns

None.

Examples

```
stapl config -handle $fp -scan-chain [list [dict create name xcvc1902
idcode 0 irlen 0 idcode2 0 mask 0] [dict create name xcvm1802  idcode 0
irlen 0 idcode2 0 mask 0]]
```

Add xcvc1902 and xcvm1802 devices to scan-chain and use the file handle returned by Tcl open command, to record stapl commands.

```
stapl config -out mystapl.stapl -scan-chain [list [dict create  name
xcvc1902 idcode 0 irlen 0 idcode2 0 mask 0] [dict create  name xcvm1802
idcode 0 irlen 0 idcode2 0 mask 0]]
```

Same as the previous example, but using the stapl file path as input,

```
instead of the file handle returned by Tcl open command.
stapl config -out mystapl.stapl -part xcvc1902
```

Add xcvc1902 device to scan-chain, using -part option.

```
stapl config -out mystapl.stapl -scan-chain [list [dict create  idcode
0x14CA8093 idcode2 1]]
```

Same as previous example, but specifying idcode and idcode2, instead of the part name.

```
stapl config -out mystapl.stapl -part [list xcvc1902 xcvm1802]
```

Add xcvc1902 and xcvm1802 devices to scan-chain, using the -part option.

```
connect
stapl config -out mystapl.stapl -scan-chain [list [dict create  name
xcvc1902 idcode 0 irlen 0 idcode2 0 mask 0]]
jtag targets -set -filter {name == "xcvc1902"}
stapl start
device program <pdipath>
stapl stop
```

The above example demonstrate the correct order for creating a stapl file for a single device on a stapl target.

```

connect
stapl config -out mystapl.stapl -scan-chain [list [dict create  name
xcvc1902 idcode 0 irlen 0 idcode2 0 mask 0] [dict create  xcvm1802 idcode
0 irlen 0 idcode2 0 mask 0]]
jtag targets -set -filter {name == "xcvc1902"}
targets -set -filter {jtag_device_name == "xcvc1902"}
stapl start
device program <pdipath>
jtag targets -set -filter {name == "xcvm1802"}
targets -set -filter {jtag_device_name == "xcvm1802"}
stapl start
device program <pdipath>
stapl stop
```

The above example demonstrate the correct order for creating a stapl file for multiple devices on a stapl target.

stapl start

Start stapl recording.

Syntax

```
stapl start
```

Start stapl recording.

Options

None.

Note(s)

- It is mandatory to call 'stapl start' before programming each device on the scan-chain, and call 'stapl stop' after programming all the devices to generate stapl data properly.

Returns

None.

stapl stop

Stop stapl recording.

Syntax

```
stapl stop
```

Stop stapl recording.

Options

None.

Note(s)

- It is mandatory to call 'stapl start' before programming each device on the scan-chain, and call 'stapl stop' after programming all the devices to generate stapl data properly.

Returns

None.

Vitis Projects

The following is a list of projects commands:

openhw

Open a hardware design.

Syntax

```
openhw <hw-proj | xsa file>
```

Open a hardware design exported from Vivado. XSA file exported from Vivado, or the hardware project created using 'createhw' command can be passed as argument.

Options

None.

Returns

If successful, this command returns nothing. Otherwise it returns an error.

Examples

```
openhw ZC702_hw_platform
```

Open the hardware project ZC702_hw_platform.

```
openhw /tmp/wrk/hw1/system.xsa
```

Open the hardware project corresponding to the system.xsa.

closehw

Close a hardware design.

Syntax

```
closehw <hw project | xsa file>
```

Close a hardware design that was opened using 'openhw' command. XSA file exported from Vivado, or the hardware project created using 'createhw' command can be passed as argument.

Options

None.

Returns

If successful, this command returns nothing. Otherwise it returns an error.

Examples

```
closehw ZC702_hw_platform
```

Close the hardware project ZC702_hw_platform.

```
closehw /tmp/wrk/hw1/system.xsa
```

Close the hardware project corresponding to the system.xsa.

getaddrmap

Get the address ranges of IP connected to processor.

Syntax

```
getaddrmap <hw spec file> <processor-instance>
```

Return the address ranges of all the IP connected to the processor in a tabular format, along with details like size and access flags of all IP.

Options

None.

Returns

If successful, this command returns the output of IPs and ranges. Otherwise it returns an error.

Examples

```
getaddrmap system.xsa ps7_cortexa9_0
```

Return the address map of peripherals connected to ps7_cortexa9_0. system.xsa is the hw specification file exported from Vivado.

getperipherals

Get a list of all peripherals in the HW design

Syntax

```
getperipherals <xsa> <processor-instance>
```

Return the list of all the peripherals in the hardware design, along with version and type.
If [processor-instance] is specified, return only a list of slave peripherals connected to that processor.

Options

None.

Returns

If successful, this command returns the list of peripherals. Otherwise it returns an error.

Examples

```
getperipherals system.xsa
```

Return a list of peripherals in the hardware design.

```
getperipherals system.xsa ps7_cortexa9_0
```

Return a list of peripherals connected to processor ps7_cortexa9_0 in the hardware design.

getprocessors

Get a list of all processors in the hardware design.

Syntax

```
getprocessors <xsa>
```

Return the list of all the processors in the hardware design

Options

None.

Returns

If successful, this command returns the list of processors. Otherwise, it returns an error.

Examples

```
getprocessors system.xsa
```

Return a list of processors in the hardware design.

repo

Get, set, or modify software repositories

Syntax

```
repo [OPTIONS]
```

Get/set the software repositories path currently used. This command is used to scan the repositories, to get the list of OS/libs/drivers/apps from repository.

Options

Option	Description
`-set <path-list>`	Set the repository path and load all the software cores available. Multiple repository paths can be specified as Tcl list.
`-get`	Get the repository path(s).
`-scan`	Scan the repositories. Used this option to scan the repositories, when some changes are done.
`-os`	Return a list of all the OS from the repositories.
`-libs`	Return a list of all the libs from the repositories.
`-drivers`	Return a list of all the drivers from the repositories.
`-apps`	Return a list of all the applications from repositories along with the following details. Supported processor - Processors for which the application can be built. Supported OS - OS for which the application can be built. Platform required - Indicates whether a platform is required to create the application. AIE applications need a platform while other applications can be created using a platform or xsa.
`-add-platforms <platforms directory>`	Add the specified directory to the platform repository.
`-remove-platforms-dir <platforms directory>`	Remove the specified directory from the platform repository.

Returns

Depends on the OPTIONS specified.

`-scan`, `-set`: Returns nothing.

`-get`: Returns the current repository path.

`-os`, `-libs`, `-drivers`, `-apps`: Returns the list of OS/libs/drivers/apps respectively.

Examples

```
repo -set <repo-path>
```

Set the repository path to the path specified by `<repo-path>`.

```
repo -os
```

Return a list of OS from the repo.

```
repo -libs
```

Return a list of libraries from the repo.

lscript

Create linker script.

Syntax

```
lscript <sub-command> [options]
```

Create a linkerscript, or perform various other operations on the linker script, based on the sub-command specified. Following sub-commands are supported. memory - List of the memories supported by the active domain. section - Lists and edit the sections available. def-mem - Returns default memory for the section type. generate - Generate a linker script. Type "help" followed by "lscript sub-command", or "lscript sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command. Refer to the sub-command help for details.

Examples

Refer to the sub-command help for details.

lscript memory

List supported memory.

Syntax

```
lscript memory [options]
```

List of the memories supported by the active domain.

Options

Option	Description
`-app <application-name>`	Name of application from workspace.
`-supported-mem`	Returns supported memory regions for each section.

Returns

List of the memories supported by the active domain in tabular format.

Examples

```
lscript memory
```

This command returns the list of memories available in the active domain.

```
lscript memory -app <application-name>
```

Returns list of memories available for application specified. This command makes the platform and domain of the specified application into the active platform and domain.

```
lscript memory -supported-mem
```

Returns the section wise supported memories.

lscript section

List the sections available.

Syntax

```
lscript section [options]
```

List, add, and edit the sections available in the active domain.

Options

Option	Description
`-app <application-name>`	Name of application from workspace.
`-name <section-name>`	Name of the section to be edited.
`-mem <memory-region>`	Name of the memory region to be used for the section.
`-size <section-size>`	Size of the section.
`-add`	Add a new section.
`-type`	Type of new section to be added. Supported types are CODE, DATA, STACK, HEAP.

Returns

List of the sections with corresponding memory and size in tabular format, when no options or args are specified. Nothing, if a section sucessfully edited or added. Error, if the section cannot be edited or added.

Examples

```
lscript section
```

List of the sections available in the active domain along with the type, size and assigned memory.

```
lscript section -app <application-name>
```

List of the sections available for application specified. This command makes the platform and domain of the specified application into the active platform and domain.

```
lscript section -name <section-name> -mem <memory-region> -size <section-
size>
```

Edit the section-name with memory and size.

```
lscript section -mem <memory-region> -size <section-size>
```

Edit all the sections with memory and size.

```
lscript section -add -name <section-name> -mem <memory-region> -size
<section-size>
-type <section-type>
```

Add a new section with section-name, memory, and size.

lscript def-mem

Returns the default memory region for the section type.

Syntax

```
lscript def-mem <memory-type>
```

Return the default memory region of the section type.

Options

Option	Description
`-app <application-name>`	Name of application from workspace.
`-code`	Return default code memory.
`-data`	Return default data memory.
`-stack`	Return default stack & heap memory.

Returns

Return the default memory region of the section type.

Examples

```
lscript def-mem -stack
```

Return default stack and heap memory-region.

```
lscript def-mem -stack -app <application-name>
```

Return default stack and heap memory region for app specified.

lscript generate

Generate a linker script.

Syntax

```
lscript generate [options]
```

Generate a linker script.

Options

Option	Description
`-app <application-name>`	Name of application from workspace.
`-name <linkerscript name>`	Name of the linker script file. The default linker script will be "newlscript.ld" if -name is not provided.
`-path <path>`	The directory where the linker script needs to be created, The default path will be pwd if -path is not provided.

Returns

Nothing.

Examples

```
lscript generate -name <linkerscript name> -path <path>
```

This command generates a linkerscript with the changes at the path provided. Otherwise, generate a default linker script with name "newlscript.ld".

```
lscript generate -app <application-name>
```

This command generates the default linker script for the application-name specified.

platform

Create, configure, list, and report platforms.

Syntax

```
platform <sub-command> [options]
```

Create a platform project, or perform various other operations on the platform project, based on the sub-command specified. Following sub-commands are supported.

- active - Set or return the active platform.
- clean - Clean platform.
- config - Configure the properties of a platform.
- create - Create/define a platform.
- generate - Build the platform.
- list - List all the platforms in workspace.
- report - Report the details of a platform.
- read - Read the platform settings from a file.
- remove - Delete the platform.
- write - Save the platform settings to a file.

Type "help" followed by "platform sub-command", or "platform sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command. Refer to the sub-command help for details.

Examples

Refer to the sub-command help for details.

platform active

Set/get active platform.

Syntax

```
platform active [platform-name]
```

Set or get the active platform. If a platform-name is specified, it is made the active platform. Otherwise, the name of active platform is returned. If no active platform exists, this command returns an empty string.

Options

None.

Returns

An empty string, if a platform is set as active or no active platform exists. The platform name, when the active platform is read.

Examples

```
platform active
```

Return the name of the active platform.

```
platform active zc702_platform
```

Set zc702_platform as active platform.

platform clean

Clean platform.

Syntax

```
platform clean
```

Clean the active platform in the workspace. This will clean all the components in platform such as FSBL, PMUFW, and so on.

Options

None.

Returns

Nothing. Build log will be printed on the console.

Examples

```
platform active zcu102
platform clean
```

Set zcu102 as the active platform and clean it.

platform config

Configure the active platform.

Syntax

```
platform config [options]
```

Configure the properties of active platform.

Options

Option	Description
-desc <description>	Add a brief description about the platform.
-updatehw <hw-spec>	Update the platform to use a new hardware specification file specified by <hw-spec>.
-samples <samples-dir>	Make the application template specified in <samples-dir> part of the platform. This option can only be used for acceleratable applications. "repo -apps <platform-name>" can be used to list the application templates available for the given platform-name.
-prebuilt-data <directory-name>	For expandable platforms, pre-generated hardware data specified in directory-name will be used for building user applications that do not contain accelerators. This will reduce the build time.
-make-local	Make the referenced SW components local to the platform.
-fsbl-target <processor-type>	Processor-type for which the existing fsbl has to be re-generated. This option is valid only for ZU+.
-create-boot-bsp	Generate boot components for the platform.
-remove-boot-bsp	Remove all the boot components generated during platform creation.
-fsbl-elf <fsbl.elf>	Prebuilt fsbl.elf to be used as boot component when "remove-boot-bsp" option is specified.
-pmufw-elf <pmufw.elf>	Prebuilt pmufw.elf to be used as boot component when "remove-boot-bsp" option is specified.
-extra-compiler-flags <param> <value>	Set extra compiler flag for the parameter with a provided value. Only FSBL and PMUFW are the supported parameters. If the value is not passed, the existing value will return.
-extra-linker-flags <param> <value>	Set extra linker flag for the parameter with a provided value. Only FSBL and PMUFW are the supported parameters. If the value is not passed, the existing value will return.
-reset-user-defined-flags <param>	Resets the extra compiler and linker flags. Only FSBL and PMUFW are the supported parameters.
-report <param>	Return the list of extra compiler and linker flags set to the given parameter. Only FSBL and PMUFW are the supported parameters.

Returns

Empty string, if the platform is configured successfully. Error string, if no platform is active or if the platform cannot be configured.

Examples

```
platform active zc702
platform config -desc "ZC702 with memory test application"
-samples /home/user/newDir
```

Make zc702 the active platform, configure the description of the platform, and make samples in the /home/user/newDir part of the platform.

```
platform config -updatehw /home/user/newdesign.xsa
```

Updates the platform project with the new XSA.

```
platform config -fsbl-target psu_cortexr5_0
```

Changes the FSBL target to psu_cortexr5_0.

```
platform config -extra-compiler-flags fsbl
```

Get the extra compiler flags. These are the flags added extra to the flags derived from the libraries, processor, and OS.

```
platform config -extra-compiler-flags fsbl "-DFSBL_DEBUG_INFO [platform
config
-extra-compiler-flags fsbl]"
```

Prepend -DFSBL_DEBUG_INFO to the compiler options, while building the fsbl application.

```
platform config -report fsbl
```

Return table of extra compiler and extra linker flags that are set for FSBL.

```
Platform config -create-boot-bsp
```

Create the boot components for the platform.

```
Platform config -create-boot-bsp -arch 32-bit
```

Create the boot components for the platform, creating FSBL in 32-bit. This is valid only for Zynq UltraScale+ MPSoC based platforms.

```
Platform config -remove-boot-bsp
```

Remove all the boot components generated during platform creation.

platform create

Create a new platform.

Syntax

```
platform create [options]
```

Create a new platform by importing hardware definition file. Platform can also be created from pre-defined hardware platforms. Supported pre-defined platforms are zc702, zcu102, zc706 and zed.

Options

Option	Description
-name <software-platform name>	Name of the software platform to be generated.
-desc <description>	Brief description about the software platform.
-hw <handoff-file>	Hardware description file to be used to create the platform.
-out <output-directory>	The directory where the software platform needs to be created. If the workspace is set, this option should not be used. Use of this option prevents the usage of platform in Vitis IDE.
-prebuilt	Mark the platform to be built from already built software artifacts. This option should be used only if you have existing software platform artifacts.
-proc <processor>	The processor to be used; the tool creates the default domain.
-arch <processor architecture>	32-bit or 64-bit, this is valid only for the A53 processor.
-samples <samples-directory>	Make the samples in <samples-directory>, part of the platform.
-os <os>	The OS to be used. The tool creates the default domain. This works in combination with -proc option.
-xpfm <platform-path>	Existing platform from which the projects have to be imported and made part of the current platform.
-no-boot-bsp	Mark the platform to build without generating boot components.
-arch <arch-type>	Processor architecture, <arch-type> can be 32 or 64 bits. This option is used to build the project with 32/64 bit toolchain.
-rp <slot-info>	Reconfigurable partition slot information for dfx flows.This option takes tcl dictionary with key-value pairs. Multiple slots can be passed as an array.

Returns

Empty string, if the platform is created successfully. Error string, if the platform cannot be created.

Examples

```
platform create -name "zcu102_test" -hw zcu102
```

Defines a software platform for a pre-defined hardware description file.

```
platform create -name "zcu102_test" -hw zcu102 -proc psu_cortexa53_0 -os
standalone
```

Defines a software platform for a pre-defined hardware desciption file. Create a default domain with standalone os running on psu_Cortexa53_0.

```
platform create -name "zcu102_32bit" -hw zcu102 -proc psu_cortexa53_0 -arch
32-bit -os standalone
```

Defines a software platform for a pre-defined hardware desciption file. Create a default domain with standalone os running on psu_Cortexa53_0 in 32-bit mode.

```
platform create -name "zcu102_test" -hw zcu102 -proc psu_cortexa53 -os
linux -arch 32-bit
```

Defines a software platform for a pre-defined hardware desciption file. Create a default domain with linux os running on psu_Cortexa53 in 32-bit.

```
platform create -xpfm /path/zc702.xpfm
```

This creates a platform project for the platform pointed by the xpfm file.

```
platform create -name "ZC702Test" -hw /path/zc702.xsa
```

Defines a software platform for a hardware desciption file.

```
platform create -name "testplat" -hw static.xsa -rp { id 1 hw ./hw.xsa
hw_emu ./hw_emu.xsa }
```

This creates a platform project with single slot DFX. User must specify path to hw XSA and hw_emu XSA.

```
platform create -name :testplat: -hw static.xsa -rp { { id 1 hw ./rp_1.xsa
hw_emu ./hw_emu.xsa } { id 2 hw ./rp_2.xsa hw_emu ./hw_emu.xsa } }
```

This creates a platform project with multi-slot DFX. The first slot is a default ReconfigurableParition. For multi-slot platforms, there are multiple hw XSAs with a slot_id for each slot and potentially multiple hw_emu XSAs or single XSA.

platform generate

Build a platform.

Syntax

```
platform generate
```

Build the active platform and add it to the repository. The platform must be created through platform create command, and must be selected as active platform before building.

Options

Option	Description
`-domains <domain-list>`	List of domains which need to be built and added to the repository. Without this option, all the domains that are part of the plafform are built.

Returns

Empty string, if the platform is generated successfully. Error string, if the platform cannot be built.

Examples

```
platform generate
```

Build the active platform and add it to repository.

```
platform generate -domains a53_standalone,r5_standalone
```

Build only a53_standalone,r5_standalone domains and add it to the repository.

platform list

List the platforms.

Syntax

List the platforms in the workspace and repository.

Options

Option	Description
`-dict`	List all the platforms for the workspace in Tcl dictionary format. Without this option, platforms are listed in tabular format.

Returns

List of platforms, or "No active platform present" string if no platforms exist.

Examples

```
platform list
```

Return a list of all the platforms in the workspace and repository in tabular format.

```
platform list -dict
```

Return a list of all the platforms in the workspace and repository in Tcl dictionary format.

platform report

Report the details of the active platform.

Syntax

```
platform report
```

Returns details such as domains and processors created in the active platform.

Options

None.

Returns

Table with details of active platform, or error string if no platforms exist.

Examples

```
platform report
```

Returns a table with details of the active platform.

platform read

Read from the platform file.

Syntax

```
platform read [platform-file]
```

Reads platform settings from the platform file and makes it available for edit. The platform file is created during the creation of platform itself and it contains all details of the platform such as hardware specification file, processor information, and so on.

Options

None.

Returns

Empty string, if the platform is read successfully. Error string, if the platform file cannot be read.

Examples

```
platform read <platform.spr>
```

Reads the platform from the platform.spr file.

platform remove

Delete a platform.

Syntax

```
platform remove <platform-name>
```

Delete the given platform. If the platform-name is not specified, the active platform is deleted.

Options

None.

Returns

Empty string, if the platform is deleted successfully. Error string, if the platform cannot be deleted.

Examples

```
platform remove zc702
```

Removes zc702 platform from the disk.

platform write

Write platform settings to a file.

Syntax

```
platform write
```

Writes the platform settings to platform.spr file. It can be read back using the "platform read" command.

Options

None.

Returns

Empty string, if the platform settings are written successfully. Error string, if the platform settings cannot be written.

Examples

```
platform write
```

Writes platform to platform.spr file.

domain

Create, configure, list and report domains.

Syntax

```
domain <sub-command> [options]
```

Create a domain, or perform various other operations on the domain, based on the sub-command specified. Following sub-commands are supported. active - Set/get the active domain. config - Configure the properties of a domain. create - Create a domain in the active platform. list - List all the domains in active platform. report - Report the details of a domain. remove - Delete a domain. Type "help" followed by "app sub-command", or "app sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command. Refer to the sub-command help for details.

Examples

Refer to the sub-command help for details.

domain active

Set/Get the active domain

Syntax

```
domain active [domain-name]
```

Set or get the active domain. If domain-name is specified, it is made the active domain. Otherwise, the name of the active domain is returned. If no active domain exists, this command returns an empty string.

Options

None.

Returns

Empty string, if a domain is set as active or no active domain exists. Domain name, when active domain is read.

Examples

```
domain active
```

Return the name of the active domain.

```
domain active test_domain
```

Set test_domain as active domain.

domain config

Configure the active domain.

Syntax

```
domain config [options]
```

Configure the properties of active domain.

Options

Option	Description
`-display-name <display name>`	Display name of the domain.
`-desc <description>`	Brief description of the domain.
`-sd-dir <location>`	For domain with Linux as OS, use pre-built Linux images from this directory, while creating the PetaLinux project. This option is valid only for Linux domains.
`-generate-bif`	Generate a standard bif for the domain. Domain report shows the location of the generated bif. This option is valid only for Linux domains.

Option	Description
`-sw-repo <repositories-list>`	List of repositories to be used to pick software components like drivers and libraries while generating this domain. Repository list should be a tcl list of software repository paths.
`-mss <mss-file>`	Use mss from specified by `<mss-file>`, instead of generating mss file for the domain.
`-readme <file-name>`	Add a README file for the domain, with boot instructions and so on.
`-inc-path <include-path>`	Additional include path which should be added while building the application created for this domain.
`-lib-path <library-path>`	Additional library search path which should be added to the linker settings of the application created for this domain.
`-sysroot <sysroot-dir>`	The Linux sysroot directory that should be added to the platform. This sysroot will be consumed during application build.
`-boot <boot-dir>`	Directory to generate components after Linux image build.
`-bif <file-name>`	Bif file used to create boot image for Linux boot.
`-qemu-args <file-name>`	File with all PS QEMU args listed. This is used to start PS QEMU.
`-pmuqemu-args <file-name>`	File with all PMC QEMU args listed. This is used to start PMU QEMU.
`-pmcqemu-args <file-name>`	File with all pmcqemu args listed. This is used to start pmcqemu.
`-qemu-data <data-dir>`	Directory which has all the files listed in file-name provided as part of qemu-args and pmuqemu-args options.

Returns

Empty string, if the domain is configured successfully. Error string, if no domain is active or if the domain cannot be configured.

Examples

```
domain config -display-name zc702_MemoryTest
-desc "Memory test application for Zynq"
```

Configure display name and description for the active domain.

```
domain config -image "/home/user/linux_image/"
```

Create PetaLinux project from pre-built Linux image.

```
domain -inc-path /path/include/ -lib-path /path/lib/
```

Adds include and library search paths to the domain's application build settings.

domain create

Create a new domain.

Syntax

```
domain create [options]
```

Create a new domain in active platform.

Options

Option	Description
`-name <domain-name>`	Name of the domain.
`-display-name <display_name>`	The name to be displayed in the report for the domain.
`-desc <description>`	Brief description of the domain.
`-proc <processor>`	Processor core to be used for creating the domain. For SMP Linux, this can be a Tcl list of processor cores.
`-arch <processor architecture>`	32-bit or 64-bit. This option is valid only for A53 processors.
`-os <os>`	OS type. Default type is standalone.
`-support-app <app-name>`	Create a domain with BSP settings needed for application specified by `<app-name>`. This option is valid only for standalone domains. The "repo -apps" command can be used to list the available application.
`-auto-generate-linux`	Generate the Linux artifacts automatically.
`-sd-dir <location>`	For domain with Linux as OS, use pre-built Linux images from this directory, while creating the PetaLinux project. This option is valid only for Linux domains.
`-sysroot <sysroot-dir>`	The Linux sysroot directory that should be added to the platform. This sysroot will be consumed during application build.

Returns

Empty string, if the domain is created successfully. Error string, if the domain cannot be created.

Examples

```
domain create -name "ZUdomain" -os standalone -proc psu_cortexa53_0
-support-app {Hello World}
```

Create a standalone domain and configure settings needed for a Hello World template application.

```
domain create -name "SMPLinux" -os linux
-proc {ps7_cortexa9_0 ps7_cortexa9_1}
```

Create a Linux domain named SMPLinux for processor cores ps7_cortexa9_0 ps7_cortexa9_1 in the active platform.

```
domain create -name a53_0_Standalone -os standalone
-proc psu_cortexa53_0 -arch 32-bit
```

Create a standalone domain for a53_0 processor for 32-bit mode.

domain list

List domains.

Syntax

```
domain list
```

List domains in the active platform.

Options

Option	Description
`-dict`	List all the domains for the active platform in Tcl dictionary format. Without this option, domains are listed in tabular format.

Returns

List of domains in the active platform, or empty string if no domains exist.

Examples

```
platform active
```

platform1

```
domain list
```

Display all the domain created in platform1 in tabular format.

```
domain list -dict
```

Display all the domain created in platform1 in Tcl dictionary format.

domain remove

Delete a domain.

Syntax

```
domain remove [domain-name]
```

Delete a domain from active platform. If a domain-name is not specified, the active domain is deleted.

Options

None.

Returns

Empty string, if the domain is deleted successfully. Error string, if the domain deletion fails.

Examples

```
domain remove test_domain
```

Removes test_domain from the active platform.

domain report

Report the details of a domain.

Syntax

```
domain report [domain-name]
```

Return details such as platform, processor core, OS, and so on. of a domain. If domain-name is not specified, details of the active domain are reported.

Options

None.

Returns

Table with details of a domain, if domain-name or active domain exists. Error string, if active domain does not exist and domain-name is not specified.

Examples

```
domain report
```

Return a table with details of the active domain.

bsp

Configure BSP settings of a baremetal domain.

Syntax

```
bsp <sub-command> [options]
```

Configure the BSP settings, including the library, driver, and OS version of a active domain, based on the sub-command specified. Following sub-commands are supported. config - Modify the configurable parameters of BSP settings. getdrivers - List IP instance and its driver. getlibs - List the libraries from BSP settings. getos - List os details from BSP settings. listparams - List the configurable parameters of os/proc/library. regenerate - Regenerate BSP sources. reload - Revert the BSP settings to the earlier saved state. write - Save the BSP edits. removelib - Remove library from bsp settings. setdriver - Sets the driver for the given IP instance. setlib - Sets the given library. setosversion - Sets version for the given OS. Type "help" followed by "bsp sub-command", or "bsp sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command. Refer to the sub-command help for details.

Examples

Refer to the sub-command help for details.

bsp config

Configure parameters of BSP settings.

Syntax

```
bsp config <param> <value>
```

Set/get/append value to the configurable parameters. If `<param>` is specified and `<value>` is not specified, return the value of the parameter. If `<param>` and `<value>` are specified, set the value of parameter. Use "bsp list-params `<-os/-proc/-driver>`" to know configurable parameters of OS/processor/driver.

Options

Option	Description
`-append <param> <value>`	Append the given value to the parameter.

Returns

Nothing, if the parameter is set/appended successfully. Current value of the paramter if `<value>` is not specified. Error string, if the parameter cannot be set/appended.

Examples

```
bsp config -append extra_compiler_flags "-pg"
```

Append -pg to extra_compiler_flags.

```
bsp config stdin
```

Return the current value of stdin.

```
bsp config stdin ps7_uart_1
```

Set stdin to ps7_uart_1.

bsp getdrivers

List drivers.

Syntax

```
bsp getdrivers
```

Return the list of drivers assigned to IP in BSP.

Options

Option	Description
`-dict`	Return the result as `<IP-name driver:version>` pairs.

Returns

Table with IP, its corresponding driver, and driver version. Empty string, if there are no IPs.

Examples

```
bsp getdrivers
```

Return the list of IPs and their drivers.

bsp getlibs

List libraries added in the BSP settings.

Syntax

```
bsp getlibs
```

Display list of libraries added in the BSP settings.

Options

Option	Description
`-dict`	Return the result as `<lib-name version>` pairs.

Returns

List of libraries. Empty string, if no libraries are added.

Examples

```
bsp getlibs
```

Return the list of libraries added in bsp settings of active domain.

bsp getos

Display OS details from BSP settings.

Syntax

```
bsp getos
```

Displays the current OS and version.

Options

Option	Description
`-dict`	Return the result as `<os-name version>` pair.

Returns

OS name and version.

Examples

```
bsp getos
```

Return OS name and version from the BSP settings of the active domain.

bsp listparams

List the configurable parameters of the BSP.

Syntax

```
bsp listparams <option>
```

List the configurable parameters of the `<option>`.

Options

Option	Description
`-lib <lib-name>`	Return the configurable parameters of the library in BSP.
`-os`	Return the configurable parameters of the OS in BSP.
`-proc`	Return the configurable parameters of the processor in BSP.

Returns

Parameter names. Empty string, if no parameters exist.

Examples

```
bsp listparams -os
```

List all the configurable parameters of OS in the BSP settings.

bsp regenerate

Regenerate BSP sources.

Syntax

```
bsp regenerate
```

Regenerate the sources with the modifications made to the BSP.

Options

None.

Returns

Nothing, if the BSP is generated successfully. Error string, if the BSP cannot be generated.

Examples

```
bsp regenerate
```

Regenerate the BSP sources with the changes to the BSP settings applied.

bsp removelib

Remove library from BSP settings.

Syntax

```
bsp removelib -name <lib-name>
```

Remove the library from BSP settings of the active domain. The library settings will come into effect only when the platform is generated. Settings can also be saved by running 'bsp write' if the user wishes to exit xsct without generating platform and revisit later.

Options

Option	Description
`-name <lib-name>`	Library to be removed from BSP settings. This is the default option, so lib-name can be directly specified as an argument without using this option.

Returns

Nothing, if the library is removed successfully. Error string, if the library cannot be removed.

Examples

```
bsp removelib -name xilffs
```

Remove xilffs library from BSP settings.

bsp setdriver

Set the driver to IP.

Syntax

```
bsp setdriver [options]
```

Set specified driver to the IP core in BSP settings of active domain.

Options

Option	Description
`-driver <driver-name>`	Driver to be assigned to an IP.
`-ip <ip-name>`	IP instance for which the driver has to be added.
`-ver <version>`	Driver version.

Returns

Nothing, if the driver is set successfully. Error string, if the driver cannot be set.

Examples

```
bsp setdriver -ip ps7_uart_1 -driver generic -ver 2.0
```

Set the generic driver for the ps7_uart_1 IP instance for the BSP.

bsp setlib

Adds the library to the BSP settings.

Syntax

```
bsp setlib [options]
```

Queues the library for addition to the active BSP. 'bsp write' will commit the queued libraries to the mss. The newly added libraries become available to the application projects after the platform is generated. If the user wants to build the platform from GUI without committing the queued libraries to mss, then the project must be cleaned first.

Options

Option	Description
`-name <lib-name>`	Library to be added to the BSP settings. This is the default option, so lib-name can be directly specified as an argument without using this option.
`-ver <version>`	Library version.

Returns

Nothing, if the library is set successfully. Error string, if the library cannot be set.

Examples

```
bsp setlib -name xilffs
```

Add the xilffs library to the BSP settings.

bsp setosversion

Set the OS version.

Syntax

```
bsp setosversion [options]
```

Set OS version in the BSP settings of the active domain. Latest version is added by default.

Options

Option	Description
`-ver <version>`	OS version.

Returns

Nothing, if the OS version is set successfully. Error string, if the OS version cannot be set.

Examples

```
bsp setosversion -ver 6.6
```

Set the OS version 6.6 in the BSP settings of the active domain.

library

Library project management

Syntax

```
library <sub-command> [options]
```

Create a library project, or perform various other operations on the library project, based on the sub-command specified. Following sub-commands are supported. build - Build the library project. clean - Clean the library project. config - Configure C/C++ build settings of the library project. create - Create a library project. list - List all the library projects in workspace. remove - Delete the library project. report - Report the details of the library project. Type "help" followed by "library sub-command", or "library sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command.

Examples

See sub-command help for examples.

library build

Build library project.

Syntax

```
library build -name <project-name>
```

Build the library project specified by `<project-name>` in the workspace. "-name" switch is optional, so `<project-name>` can be specified directly, without using -name.

Options

Option	Description
`-name <project-name>`	Name of the library project to be built.

Returns

Nothing, if the library project is built successfully. Error string, if the library project build fails.

Examples

```
library build -name lib1
```

Build lib1 library project.

library clean

Clean library project.

Syntax

```
library clean -name <project-name>
```

Clean the library project specified by `<project-name>` in the workspace. "-name" switch is optional, so `<project-name>` can be specified directly, without using -name.

Options

Option	Description
`-name <project-name>`	Name of the library project to be clean built.

Returns

Nothing, if the library project is cleaned successfully. Error string, if the library project build clean fails.

Examples

```
library clean -name lib1
```

Clean lib1 library project.

library create

Create a library project.

Syntax

```
library create -name <project-name> -type <library-type> -platform
<platform>
```

-domain <domain> -sysproj <system-project> Create a library project using an existing platform, and domain. If <platform>, <domain>, and <sys-config> are not specified, the active platform and domain are used for creating the library project. For creating a library project and adding it to an existing system project, refer to the next use case.

```
library create -name <project-name> -type <library-type> -sysproj <system-
project>
```

-domain <domain> Create a library project for domain specified by <domain> and add it to system project specified by <system-project>. If <system-project> exists, platform corresponding to this system project are used for creating the library project. If <domain> is not specified, the active domain is used.

Options

Option	Description
-name <project-name>	Project name that should be created.
-type <library-type>	<library-type> can be 'static' or 'shared'.
-platform <platform-name>	Name of the platform. Use "repo -platforms" to list available pre-defined platforms.
-domain <domain-name>	Name of the domain. Use "platform report <platform-name>" to list the available domains in a platform.
-sysproj <system-project>	Name of the system project. Use "sysproj list" to know the available system projects in the workspace.

Returns

Nothing, if the library project is created successfully. Error string, if the library project creation fails.

Examples

```
library create -name lib1 -type static -platform zcu102 -domain
a53_standalone
```

Create a static library project with name 'lib1', for the platform zcu102, which has a domain named a53_standalone domain.

```
library create -name lib2 -type shared -sysproj test_system -domain
test_domain
```

Create shared library project with name 'lib2' and add it to system project test_system.

library list

List library projects.

Syntax

List all library projects in the workspace.

Options

None.

Returns

List of library projects in the workspace. If no library projects exist, an empty string is returned.

Examples

```
library list
```

Lists all the library projects in the workspace.

library remove

Delete library project.

Syntax

```
library remove [options] <project-name>
```

Delete a library project from the workspace.

Options

None.

Returns

Nothing, if the library project is deleted successfully. Error string, if the library project deletion fails.

Examples

```
library remove lib1
```

Removes lib1 from workspace.

library report

Report details of the library project.

Syntax

```
library report <project-name>
```

Return details such as the platform, domain, and so on of the library project.

Options

None.

Returns

Details of the library project, or error string, if library project does not exist.

Examples

```
app report lib1
```

Return all the details of library lib1.

checkvalidrmxsa

Check if RM XSA is suitable for static XSA.

Syntax

```
checkvalidrmxsa -hw <static hw spec file> -rm-hw <rm hw spec file>
```

To check if the RM XSA is suitable to work with the static hardware XSA.

Options

None.

Returns

If successful, returns true if the RM hardware XSA is a fit for the static hardware XSA. Returns false if not. Otherwise, it returns an error.

Examples

```
checkvalidrmxsa -hw static.xsa -rm-hw rm.xsa
```

Returns true if RM XSA can be used along with the static XSA.

isstaticxsa

Check if hardware design is a static XSA.

Syntax

```
isstaticxsa <hw spec file>
```

Checks if the hardware design is a static XSA.

Options

None.

Returns

If successful, returns true if hardware design is static, returns false if hardware design is not static. Otherwise, it returns an error.

Examples

```
isstaticxsa static.xsa
```

Returns true if XSA is static.

ishwexpandable

Check if hardware design is expandable.

Syntax

```
ishwexpandable <hw spec file>
```

Checks if the hardware design is expandable or fixed.

Options

None.

Returns

If successful, returns true if hardware design is expandable/extensible, returns false if hardware design is fixed. Otherwise, it returns an error.

Examples

```
ishwexpandable system.xsa
```

Returns true if XSA is expandable/extensible.

createdts

Creates device tree.

Syntax

```
createdts [options]
```

Create a device tree for the hardware definition file.

Options

Option	Description
-platform-name <software-platform name>	Name of the software platform to be generated.
-board <board name>	Board name for device tree to be generated. Board names available at <DTG Repo>/device_tree/data/kernel_dtsi.
-hw <handoff-file>	Hardware description file to be used to create the device tree.
-out <output-directory>	The directory where the software platform needs to be created. Workspace will be default directory, if this option is not specified.
-local-repo <directory location>	Location of the directory were bsp for git repo is available. Device tree repo will be cloned from git, if this option is not specified.
-git-url <Git URL>	Git URL of the dtg repo to be cloned. Default repo is https://github.com/Xilinx/device-tree-xlnx.git.
-git-branch <Git Branch>	Git branch to be checked out. 'xlnx_rel_v<Vitis-release>' is selected by default.
-zocl	Set zocl flag to enable zocl driver support, default set to False. zocl should only be used when the designs are PL enabled. Only master and xlnx_rel_v2021.2 branch supports zocl property.

Option	Description
-overlay	Set overlay flag to enable device-tree overlay support, default set to False.
-dtsi <custom-dtsi-file list>	Include custom-dtsi file in the device tree, if specified. The filepaths must be in the list format.
-compile	Specify this option to compile the generated dts to create dtb. If this option is not specified, users can manually use dts to compile dtb. For example, dtc -I dts -O dtb -o <file_name>.dtb <file_name>.dts Compile dts(device tree source) or dtsi(device tree source include) files. dtc -I dts -O dtb -f <file_name>.dts -o <file_name>.dtb Convert dts(device tree source) to dtb(device tree blob). dtc -I dtb -O dts -f <file_name>.dtb -o <file_name>.dts Convert dtb(device tree blob) to dts(device tree source).
-update	Set update flag to enable existing device tree platform to update with new xsa.

Note(s)

- This command is a shortcut for creating a device tree domain and generating the device tree. It clones the device tree repo, creates a platform with device_tree as OS, and configures and generates the platform to create dts. -zocl should only be used when the designs are PL enabled. Only master and xlnx_rel_v2021.2 branch supports zocl property. Git 1.5.4 or later is required to avoid any issues with the git commands used by the createdts command.

Returns

None.

Examples

```
createdts -hw zcu102.xsa -platform-name my_devicetree
```

Create a device tree for the handoff-file with default repo as "https://github.com/Xilinx/device-tree-xlnx.git" and default branch as "xlnx_rel_v<Vitis-release>".

```
createdts -hw zcu102.xsa -platform-name my_devicetree -git-url <Git URL>
-git-branch <Git Branch>
```

Create a device tree for the handoff-file with user repo as repo mentioned in <Git URL> and user branch as <Git Branch>.

```
createdts -hw zc702.xsa -platform-name my_devicetree
-local-repo /my_local_git_repo
```

Create a device tree for the handoff-file and use the local repo.

```
createdts -hw vck190.xsa -platform-name my_devicetree
-out /device-tree_output_directory
```

Create a device tree at the out directory specified by device-tre output directory.

```
createdts -hw zcu102.xsa -platform-name my_devicetree -overlay
-zocl -compile
```

Create device tree for the handoff-file with overlay and zocl node. Compile flag compiles the device tree blob file from the DTS.

```
createdts -hw zcu102.xsa -platform-name my_devicetree -board <Board Name>
```

Creates a device tree adding board value to the library, Board names available at `<DTG Repo>`/device_tree/data/kernel_dtsi.

```
createdts -update -hw newdesign.xsa
```

Updates existing device tree platform with new XSA.

```
createdts -hw vck190 -platform-name vck190 -out <out_dir>
-dtsi [list path/system-conf.dtsi path/system-user.dtsi]
```

Create device tree with custom-dtsi-files included.

setws

Set Vitis workspace

Syntax

```
setws [OPTIONS] [path]
```

Set Vitis workspace to `<path>`, for creating projects. If `<path>` does not exist, then the directory is created. If `<path>` is not specified, then current directory is used.

Options

Option	Description
`-switch <path>`	Close existing workspace and switch to new workspace.

Returns

Nothing if the workspace is set successfully. Error string, if the path specified is a file.

Examples

```
setws /tmp/wrk/wksp1
```

Set the current workspace to /tmp/wrk/wksp1.

```
setws -switch /tmp/wrk/wksp2
```

Close the current workspace and switch to new workspace /tmp/wrk/wksp2.

getws

Get Vitis workspace.

Syntax

```
getws
```

Return the current vitis workspace.

Returns

Current workspace.

app

Application project management.

Syntax

```
app <sub-command> [options]
```

Create an application project, or perform various other operations on the application project, based on `<sub-command>` specified. Following sub-commands are supported. build - Build the application project. clean - Clean the application project. config - Configure C/C++ build settings of the application project. create - Create an application project. list - List all the application projects in workspace. remove - Delete the application project. report - Report the details of the application project. switch - Switch application project to refer another platform. Type "help" followed by "app sub-command", or "app sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command. Refer to the sub-command help for details.

Examples

Refer to the sub-command help for examples.

app build

Build application.

Syntax

```
app build -name <app-name>
```

Build the application specified by `<app-name>` in the workspace. "-name" switch is optional, so `<app-name>` can be specified directly, without using -name.

Options

Option	Description
`-name <app-name>`	Name of the application to be built.
`-all`	Option to Build all the application projects.

Returns

Nothing. Build log will be printed on the console.

Examples

```
app build -name helloworld
```

Build Hello World application.

```
app build -all
```

Build all the application projects in the workspace.

app clean

Clean application.

Syntax

```
app clean -name <app-name>
```

Clean the application specified by `<app-name>` in the workspace. "-name" switch is optional, so `<app-name>` can be specified directly, without using -name.

Options

Option	Description
`-name <app-name>`	Name of the application to be clean built.

Returns

Nothing. Build log will be printed on the console.

Examples

```
app clean -name helloworld
```

Clean Hello World application.

app config

Configure C/C++ build settings of the application.

Syntax

Configure C/C++ build settings for the specified application. Following settings can be configured for applications: assembler-flags : Miscellaneous flags for assembler build-config : Get/set build configuration compiler-misc : Compiler miscellaneous flags compiler-optimization : Optimization level define-compiler-symbols : Define symbols. Ex. MYSYMBOL include-path : Include path for header files libraries : Libraries to be added while linking library-search-path : Search path for the libraries added linker-misc : Linker miscellaneous flags linker-script : Linker script for linking undef-compiler-symbols : Undefine symbols. Ex. MYSYMBOL

```
app config -name <app-name> <param-name>
```

Get the value of configuration parameter `<param-name>` for the application specified by `<app-name>`.

```
app config [OPTIONS] -name <app-name> <param-name> <value>
```

Set/modify/remove the value of configuration parameter `<param-name>` for the application specified by `<app-name>`.

Options

Option	Description
`-name`	Name of the application.
`-set`	Set the configuration parameter value to new `<value>`.
`-get`	Get the configuration parameter value.

Option	Description
`-add`	Append the new `<value>` to configuration parameter value. Add option is not supported for compiler-optimization
`-info`	Displays more information like possible values and possible operations about the configuration parameter. A parameter name must be specified when this option is used.
`-remove`	Remove `<value>` from the configuration parameter value. Remove option is not supported for assembler-flags, build-config, compiler-misc, compiler-optimization, linker-misc, and linker-script.

Returns

Depends on the arguments specified. `<none>` List of parameters available for configuration and description of each parameter.

`<parameter name>`: Parameter value, or error, if unsupported parameter is specified.

`<parameter name> <paramater value>`: Nothing if the value is set successfully, or error, if unsupported parameter is specified.

Examples

```
app config -name test build-config
```

Return the current build configuration for the application named test.

```
app config -name test define-compiler-symbols FSBL_DEBUG_INFO
```

Add -DFSBL_DEBUG_INFO to the compiler options, while building the test application.

```
app config -name test -remove define-compiler-symbols FSBL_DEBUG_INFO
```

Remove -DFSBL_DEBUG_INFO from the compiler options, while building the test application.

```
app config -name test -set compiler-misc {-c -fmessage-length=0 -MT"$@"}
```

Set {-c -fmessage-length=0 -MT"$@"} as compiler miscellaneous flags for the test application. app config -name test -append compiler-misc {-pg} Add {-pg} to compiler miscellaneous flags for the test application.

```
app config -name test -info compiler-optimization
```

Changing the compiler-optimization in the compiler options of an application.

```
app config -name test -set compiler-optimization {Optimize for size (-Os)}
```

Display more information about possible values and default values for compiler optimization level.

app create

Create an application.

Syntax

```
app create [options] -platform <platform> -domain <domain>
```

-sysproj <system-project> Create an application using an existing platform and domain, and add it to a system project. If <platform> and <domain> are not specified, then active platform and domain are used for creating the application. If <system-project> is not specified, then a system project is created with name appname_system. For creating applications and adding them to existing system project, refer to next use case. Supported options are: -name, -template.

```
app create [options] -sysproj <system-project> -domain <domain>
```

Create an application for domain specified by <domain> and add it to system project specified by <system-project>. If <system-project> exists, platform corresponding to this system project are used for creating the application. If <domain> is not specified, then active domain is used. Supported options are: -name, -template.

```
app create [options] -hw <hw-spec> -proc <proc-instance>
```

Create an application for processor core specified <proc-instance> in HW platform specified by <hw-spec>. Supported options are: -name, -template, -os, -lang, -arch.

Options

Option	Description
-name <application-name>	Name of the application to be created.
-platform <platform-name>	Name of the platform. Use "repo -platforms" to list available pre-defined platforms.
-domain <domain-name>	Name of the domain. Use "platform report <platform-name>" to list the available system configurations in a platform.
-hw <hw-spec>	HW specification file exported from Vivado (XSA).
-sysproj <system-project>	Name of the system project. Use "sysproj list" to know available system projects in the workspace.
-proc <processor>	Processor core for which the application should be created.
-template <application template>	Name of the template application. Default is "Hello World". Use "repo -apps" to list available template applications.
-os <os-name>	OS type. Default type is standalone.
-lang <programming language>	Programming language can be c or c++.
-arch <arch-type>	Processor architecture, <arch-type> can be 32 or 64 bits. This option is used to build the project with 32/64 bit toolchain.

Returns

Nothing, if the application is created successfully. Error string, if the application creation fails.

Examples

```
app create -name test -platform zcu102 -domain a53_standalone
```

Create Hello World application named test, for the platform zcu102, with a domain named a53_standalone.

```
app create -name zqfsbl -hw zc702 -proc ps7_cortexa9_0 -os standalone
-template "Zynq FSBL"
```

Create Zynq FSBL application named zqfsbl for ps7_cortexa9_0 processor core, in zc702 HW platform.

```
app create -name memtest -hw /path/zc702.xsa -proc ps7_cortexa9_0 -os
standalone
-template "Memory Tests"
```

Create Memory Test application named memtest for ps7_cortexa9_0 processor core, in zc702.xsa HW platform.

```
app create -name test -sysproj test_system -domain test_domain
```

Create Hello World application project with name test and add it to system project test_system.

app list

List applications.

Syntax

```
app list
```

List all applications for in the workspace.

Options

Option	Description
`-dict`	List all the applications for the workspace in Tcl dictionary format. Without this option, applications are listed in tabular format.

Returns

List of applications in the workspace. If no applications exist, "No application exist" string is returned.

Examples

```
app list
```

Lists all the applications in the workspace in tabular format.

```
app list -dict
```

Lists all the applications in the workspace in Tcl dictionary format.

app remove

Delete application.

Syntax

```
app remove <app-name>
```

Delete an application from the workspace.

Options

None.

Returns

Nothing, if the application is deleted successfully. Error string, if the application deletion fails.

Examples

```
app remove zynqapp
```

Removes zynqapp from workspace.

app report

Report details of the application.

Syntax

```
app report <app-name>
```

Return details such as the platform, domain, processor core, and OS of an application.

Options

None.

Returns

Details of the application, or error string, if application does not exist.

Examples

```
app report test
```

Return all the details of application test.

app switch

Switch the application to use another domain/platform.

Syntax

```
app switch -name <app-name> -platform <platform-name> -domain <domain-name>
```

Switch the application to use another platform and domain. If the domain name is not specified, application will be moved to the first domain which is created for the same processor as current domain. This option is supported if there is only one application under this platform.

```
app switch -name <app-name> -domain <domain-name>
```

Switch the application to use another domain within the same platform. New domain should be created for the same processor as current domain.

Options

Option	Description
`-name <application-name>`	Name of the application to be switched.
`-platform <platform-name>`	Name of the new Platform. Use "platform -list" to list the available platforms.
`-domain <domain-name>`	Name of the new domain. Use "domain -list" to list avaliable domain in the active platform.

Returns

Nothing if application is switched successfully, or error string, if given platform project does not exist or given platform project does not have valid domain.

Examples

```
app switch -name helloworld -platform zcu102
```

Switch the Hello World application to use zcu102 platform.

sysproj

System project management.

Syntax

```
sysproj <sub-command> [options]
```

Build, list and report system project, based on `<sub-command>` specified. Following sub-commands are supported. build - Build the system project. clean - Clean the system project. list - List all system projects in workspace. remove - Delete the system project. report - Report the details of the system project. Type "help" followed by "sysproj sub-command", or "sysproj sub-command" followed by "-help" for more details.

Options

None.

Returns

Depends on the sub-command.

Examples

See sub-command help for examples.

sysproj build

Build system project.

Syntax

```
sysproj build -name <sysproj-name>
```

Build the application specified by `<sysproj-name>` in the workspace. "-name" switch is optional, so `<sysproj-name>` can be specified directly, without using -name.

Options

Option	Description
`-name <sysproj-name>`	Name of the system project to be built.
`-all`	Option to build all the system projects.

Examples

```
sysproj build -name helloworld_system
```

Build the system project specified.

```
sysproj build -all
```

Build all the system projects in the workspace.

sysproj clean

Clean application.

Syntax

```
sysproj clean -name <app-name>
```

Clean the application specified by `<sysproj-name>` in the workspace. "-name" switch is optional, so `<sysproj-name>` can be specified directly, without using -name.

Options

Option	Description
`-name <sysproj-name>`	Name of the application to be clean built.

Returns

Nothing, if the application is cleaned suceessfully. Error string, if the application build clean fails.

Examples

```
sysproj clean -name helloworld_system
```

Clean-build the system project specified.

sysproj list

List system projects.

Syntax

```
sysproj list
```

List all system projects in the workspace.

Options

None.

Returns

List of system projects in the workspace. If no system project exist, an empty string is returned.

Examples

```
sysproj list
```

List all system projects in the workspace.

sysproj remove

Delete system project.

Syntax

```
sysproj remove [options]
```

Delete a system project from the workspace.

Options

None.

Returns

Nothing, if the system project is deleted successfully. Error string, if the system project deletion fails.

Examples

```
sysproj remove test_system
```

Delete test_system from workspace.

sysproj report

Report details of the system project.

Syntax

```
sysproj report <sysproj-name>
```

Return the details such as the platform and domain of a system project.

Options

None.

Returns

Details of the system project, or error string, if system project does not exist.

Examples

```
sysproj report test_system
```

Return all the details of the system project test_system.

importprojects

Import projects to workspace.

Syntax

```
importprojects <path>
```

Import all the Vitis projects from `<path>` to workspace.

Returns

Nothing, if the projects are imported successfully. Error string, if project path is not specified or if the projects cannot be imported.

Examples

```
importprojects /tmp/wrk/wksp1/hello1
```

Import Vitis project(s) into the current workspace.

importsources

Import sources to an application project.

Syntax

```
importsources [OPTIONS]
```

Import sources from a path to application project in workspace.

Options

Option	Description
`-name <project-name>`	Application Project to which the sources should be imported.

Option	Description
`-path <source-path>`	Path from which the source files should be imported. If `<source-path>` is a file, it is imported to application project. If `<source-path>` is a directory, all the files/ sub-directories from the `<source-path>` are imported to application project. All existing source files will be overwritten in the application, and new files will be copied. Linker script will not be copied to the application directory, unless -linker-script option is used.
`-soft-link`	Links the sources from source-path and does not copy the source.
`-target-path <dir-path>`	Directory to which the sources have to be linked or copied. If target-path option is not used, source files will be linked or copied to "src" directory.
`-linker-script`	Copies the linker script as well.

Returns

Nothing, if the project sources are imported successfully. Error string, if invalid options are used or if the project sources cannot be read/imported.

Examples

```
importsources -name hello1 -path /tmp/wrk/wksp2/hello2
```

Import the 'hello2' project sources to 'hello1' application project without the linker script.

```
importsources -name hello1 -path /tmp/wrk/wksp2/hello2 -linker-script
```

Import the 'hello2' project sources to 'hello1' application project along with the linker script.

```
importsources -name hello1 -path /tmp/wrk/wksp2/hello_app -soft-link
```

Create a soft-link to hello1 application project from hello_app application project.

toolchain

Set or get toolchain used for building projects.

Syntax

```
toolchain
```

Return a list of available toolchains and supported processor types.

```
toolchain <processor-type>
```

Get the current toolchain for `<processor-type>`.

```
toolchain <processor-type> <tool-chain>
```

Set the <toolchain> for <processor-type>. Any new projects created will use the new toolchain during build.

Returns

Depends on the arguments specified.

<none>: List of available toolchains and supported processor types.

<processor-type>: Current toolchain for processor-type.

<processor-type> <tool-chain>: Nothing if the tool-chain is set, or error, if unsupported tool-chain is specified.

XSDB Use Cases

XSDB can be used in various scenarios in the development, debugging, verification, and deployment cycles. XSDB inherits high-level, interpreted, and dynamic programming features from Tcl, which makes the programming simple and powerful.

XSDB can inter-operate the workspace together with the AMD Vitis™ IDE. When creating and managing projects, XSDB launches the Vitis IDE in the background. XSDB workspaces can be seamlessly used with the Vitis IDE and vice versa. When you are working in the Vitis IDE, equivalent XSDB commands is printed in the console in most use cases. This can help you create scripts for batching and automation when actions need to be executed repeatedly.

***Note*:** At any point in time, a workspace can either be used only from Vitis IDE or XSDB.

Common Use Cases

- **Checking the JTAG status of the board:** In the new board bring-up phase, after verifying the power circuits, the first job for hardware verification is to test the JTAG status; checking whether the FPGA or SoC device can be scanned, and whether the processors can be found properly. XSDB can do this job with JTAG access and target connection management commands such as 'jtag tagrets', 'connect', and 'targets'. If you suspect that the board is in an abnormal status and you need to check the basic hardware, it is also recommended to check the JTAG and processor status.
- **Initializing the board with a single script through JTAG:** In some debugging cases (for example, debugging a PL module that needs a PS generated clock), the PS simply needs to be initialized into a certain status. Running customized initialization scripts can be faster and more lightweight than launching runs with the Vitis IDE. The Vitis IDE shows the equivalent XSDB debug commands in the console. To repeat an initialization cycle easily, copy these commands into a Tcl file and use XSDB to execute this Tcl script.
- **Loading U-Boot with a single script through JTAG:** If you need to customize U-Boot, the easiest way to test and iterate is to use XSDB to initialize the board, load the U-Boot binary into DDR, and run it. This can be executed on the fly. Otherwise, you might have to package the `boot.bin` file and write it to an SD card or the flash memory every time you update the code.

- **Reading and writing registers with or without applications:** When debugging peripherals or their drivers, the status of the peripheral registers is important. The status can be read from XSDB or it can be viewed in the Vitis IDE memory view. Using XSDB commands to read and write registers is quick and lightweight. The register read and write commands can be written into a script to automate repeated processes. You can also save the register values into a file for comparison.

Changing Compiler Options of an Application Project

An example XSDB session that demonstrates creating an empty application for Cortex®-A53 processor, by adding the compiler option `-std=c99` is as follows.

```
setws /tmp/wrk/workspace
app create -name test_a53 -hw /tmp/wrk/system.xsa -os standalone -proc
psu_cortexa53_0 -template {Empty Application(C)}
importsources -name test_a53 -path /tmp/sources/
app config -name test_a53 -add compiler-misc {-std=c99}
app build -name test_a53
```

Creating an Application Project Using an Application Template (Zynq UltraScale+ MPSoC FSBL)

The following is an example XSDB session that demonstrates creating a FSBL project for a Cortex-A53 processor.

Note: Creating an application project creates a BSP project by adding the necessary libraries and setting compiler options automatically. `FSBL_DEBUG_DETAILED` symbol is added to FSBL for debug messages.

```
setws /tmp/wrk/workspace
app create -name a53_fsbl -hw /tmp/wrk/system.xsa -os standalone -proc
psu_cortexa53_0 -template {Zynq MP FSBL}
app config -name a53_fsbl define-compiler-symbols {FSBL_DEBUG_INFO}
app build -name a53_fsbl
```

Creating an FSBL Application Project Using Manually Created Domain (Zynq UltraScale+ MPSoC FSBL)

The following is an example XSDB session that demonstrates creating a FSBL project for a Cortex-A53 processor by manually creating platform, domain and application. Configuration option zynqmp_fsbl_bsp is set for FSBL compiler optimization options.

```
setws /tmp/wrk/workspace
platform create -name HW1 -hw zcu102 -no-boot-bsp
domain create -name A53_Standalone -os standalone -proc psu_cortexa53_0
domain active A53_Standalone
bsp setlib -name xilffs
bsp setlib -name xilsecure
bsp setlib -name xilpm
bsp config zynqmp_fsbl_bsp true

platform generate
app create -name a53_fsbl -platform HW1 -template "Zynq MP FSBL" -domain
A53_Standalone -lang c
app build -name a53_fsbl
```

Creating a Bootable Image and Program the Flash

An example XSDB session that demonstrates the creation of a "Hello World" application is shown in the following snippet. It also shows the creation of a bootable image using the applications along with bitstream by building the system project and programming the image onto the flash.

***Note*:** The Vitis environment creates a platform project and system project when an application project is created. The platform project includes boot components such as FSBL, which are required for initializing a device. This example assumes that you are using the ZC702 board, and uses `-flash_type qspi_single` as an option with `program_flash`.

```
setws /tmp/wrk/workspace
app create -name a9_fsbl -hw /tmp/wrk/system.xsa -os standalone -proc
ps7_cortexa9_0 -template {Hello World}
app build -name a9_hello
# Build the system project. This builds the platform project to generate
fsbl.elf
# and creates a bif file and runs Bootgen to create a boot image (BOOT.BIN)
sysproj build -name a9_hello_system
```

```
# Modify the bif and run Bootgen if needed
# exec bootgen -arch zynq -image output.bif -w -o /tmp/wrk/BOOT.bin
# Program the flash and verify the flash device
exec program_flash -f /tmp/wrk/BOOT.bin -flash_type qspi_single
-blank_check -verify -cable type xilinx_tcf url tcp:localhost:3121
```

Debugging a Program Already Running on the Target

System Debugger Command-line Interface (XSDB) can be used to debug a program which is already running on the target (for example, booting from flash). Connect to the target and set the symbol file for the program running on the target. This method can also be used to debug Linux kernel booting from flash. For best results, the code running on the target should be compiled with the debug information.

The following is an example of debugging a program already running on the target. For demo purpose, the program has been stopped at `main()`, before this example session.

```
# Connect to the hw_server

xsdb% conn -url TCP:xhdbfarmc7:3121
tcfchan#0
xsdb% Info: Arm Cortex-A9 MPCore #0 (target 2) Stopped at 0x1005a4
(Hardware Breakpoint)
xsdb% Info: Arm Cortex-A9 MPCore #1 (target 3) Stopped at 0xfffffe18
(Suspended)

# Select the target on which the program is running and specify the symbol
file using the
# memmap command

xsdb% targets 2
xsdb% memmap -file dhrystone/Debug/dhrystone.elf

# When the symbol file is specified, the debugger maps the code on the
target to the symbol
# file. bt command can be used to see the back trace. Further debug is
possible, as shown in
# the first example

xsdb% bt
    0  0x1005a4 main(): ../src/dhry_1.c, line 79
    1  0x1022d8 _start()+88
    2  unknown-pc
```

Debugging Applications on Zynq UltraScale+ MPSoC

***Note*:** For simplicity, this help page assumes that AMD Zynq™ UltraScale+™ MPSoC boots up in JTAG bootmode. The flow described here can be applied to other boot modes too, with minor changes.

When Zynq UltraScale+ MPSoC boots up JTAG bootmode, all the Cortex®-A53 and Cortex®-R5F cores are held in reset. Users must clear resets on each core, before debugging on these cores. The `rst` command in XSDB can be used to clear the resets. `rst -processor` clears reset on an individual processor core. `rst -cores` clears resets on all the processor cores in the group (APU or RPU), of which the current target is a child. For example, when Cortex-A53 #0 is the current target, `rst -cores` clears resets on all the Cortex-A53 cores in APU.

Below is an example XSDB session that demonstrates standalone application debug on Cortex-A53 #0 core on Zynq UltraScale+ MPSoC.

***Note*:** Similar steps can be used for debugging applications on Cortex-R5F cores and also on Cortex-A53 cores in 32 bit mode. However, the Cortex-A53 cores must be put in 32 bit mode, before debugging the applications. This should be done after POR and before the Cortex-A53 resets are cleared.

```
#connect to remote hw_server by specifying its url.
If the hardware is connected to a local machine,-url option and the <url>
are not needed. connect command returns the channel ID of the connection

xsdb% connect -url TCP:xhdbfarmc7:3121 -symbols
tcfchan#0

# List available targets and select a target through its id.
The targets are assigned IDs as they are discovered on the Jtag chain,
so the IDs can change from session to session.
For non-interactive usage, -filter option can be used to select a target,
instead of selecting the target through its ID

xsdb% targets
  1  PS TAP
     2  PMU
        3  MicroBlaze PMU (Sleeping. No clock)
     4  PL
  5  PSU
     6  RPU (Reset)
        7  Cortex-R5 #0 (RPU Reset)
        8  Cortex-R5 #1 (RPU Reset)
     9  APU (L2 Cache Reset)
       10  Cortex-A53 #0 (APU Reset)
       11  Cortex-A53 #1 (APU Reset)
       12  Cortex-A53 #2 (APU Reset)
       13  Cortex-A53 #3 (APU Reset)
xsdb% targets 5

# Configure the FPGA. When the active target is not a FPGA device,
the first FPGA device is configured

xsdb% fpga ZCU102_HwPlatform/design_1_wrapper.bit
100%    36MB   1.8MB/s  00:24
```

```
# Source the psu_init.tcl script and run psu_init command to initialize PS
xsdb% source ZCU102_HwPlatform/psu_init.tcl
xsdb% psu_init

# PS-PL power isolation must be removed and PL reset must be toggled,
before the PL address space can be accessed

# Some delay is needed between these steps

xsdb% after 1000
xsdb% psu_ps_pl_isolation_removal
xsdb% after 1000
xsdb% psu_ps_pl_reset_config

# Select A53 #0 and clear its reset

# To debug 32 bit applications on A53, A53 core must be configured
to boot in 32 bit mode, before the resets are cleared

# 32 bit mode can be enabled through CONFIG_0 register in APU module.
See ZynqMP TRM for details about this register

xsdb% targets 10
xsdb% rst -processor

# Download the application program

xsdb% dow dhrystone/Debug/dhrystone.elf
Downloading Program -- dhrystone/Debug/dhrystone.elf
                section, .text: 0xfffc0000 - 0xfffd52c3
                section, .init: 0xfffd5300 - 0xfffd5333
                section, .fini: 0xfffd5340 - 0xfffd5373
                section, .note.gnu.build-id: 0xfffd5374 - 0xfffd5397
                section, .rodata: 0xfffd5398 - 0xfffd6007
                section, .rodata1: 0xfffd6008 - 0xfffd603f
                section, .data: 0xfffd6040 - 0xfffd71ff
                section, .eh_frame: 0xfffd7200 - 0xfffd7203
                section, .mmu_tbl0: 0xfffd8000 - 0xfffd800f
                section, .mmu_tbl1: 0xfffd9000 - 0xfffdafff
                section, .mmu_tbl2: 0xfffdb000 - 0xfffdefff
                section, .init_array: 0xfffdf000 - 0xfffdf007
                section, .fini_array: 0xfffdf008 - 0xfffdf047
                section, .sdata: 0xfffdf048 - 0xfffdf07f
                section, .bss: 0xfffdf080 - 0xfffe197f
                section, .heap: 0xfffe1980 - 0xfffe397f
                section, .stack: 0xfffe3980 - 0xfffe697f
100%    0MB   0.4MB/s  00:00
Setting PC to Program Start Address 0xfffc0000
Successfully downloaded dhrystone/Debug/dhrystone.elf

# Set a breakpoint at main()
xsdb% bpadd -addr &main
0

# Resume the processor core
xsdb% con

# Info message is displayed when the core hits the breakpoint
Info: Cortex-A53 #0 (target 10) Running
xsdb% Info: Cortex-A53 #0 (target 10) Stopped at 0xfffc0d5c (Breakpoint)

# Registers can be viewed when the core is stopped
```

```
xsdb% rrd
  r0: 0000000000000000    r1: 0000000000000000    r2: 0000000000000000
  r3: 0000000000000004    r4: 000000000000000f    r5: 00000000ffffffff
  r6: 000000000000001c    r7: 0000000000000002    r8: 00000000ffffffff
  r9: 0000000000000000   r10: 0000000000000000   r11: 0000000000000000
 r12: 0000000000000000   r13: 0000000000000000   r14: 0000000000000000
 r15: 0000000000000000   r16: 0000000000000000   r17: 0000000000000000
 r18: 0000000000000000   r19: 0000000000000000   r20: 0000000000000000
 r21: 0000000000000000   r22: 0000000000000000   r23: 0000000000000000
 r24: 0000000000000000   r25: 0000000000000000   r26: 0000000000000000
 r27: 0000000000000000   r28: 0000000000000000   r29: 0000000000000000
 r30: 00000000fffc1f4c    sp: 00000000fffe5980    pc: 00000000fffc0d5c
cpsr:         600002cd   vfp                     sys

# Local variables can be viewed
xsdb% locals
Int_1_Loc       : 1113232
Int_2_Loc       : 30
Int_3_Loc       : 0
Ch_Index        : 0
Enum_Loc        : 0
Str_1_Loc       : char[31]
Str_2_Loc       : char[31]
Run_Index       : 1061232
Number_Of_Runs  : 2

# Local variable value can be modified
xsdb% locals Number_Of_Runs 100
xsdb% locals Number_Of_Runs
Number_Of_Runs  : 100

# Global variables and be displayed, and its value can be modified
xsdb% print Int_Glob
Int_Glob  : 0
xsdb% print -set Int_Glob 23
xsdb% print Int_Glob
Int_Glob  : 23

# Expressions can be evaluated and its value can be displayed
xsdb% print Int_Glob + 1 * 2
Int_Glob + 1 * 2  : 25

# Step over a line of source code
xsdb% nxt
Info: Cortex-A53 #0 (target 10) Stopped at 0xfffc0d64 (Step)

# View stack trace
xsdb% bt
    0  0xfffc0d64 main()+8: ../src/dhry_1.c, line 79
    1  0xfffc1f4c _startup()+84: xil-crt0.S, line 110
```

***Note*:** If the `.elf` file is not accessible from the remote machine on which the server is running, the `xsdb% connect -url TCP:xhdbfarmc7:3121` command should be appended with the `-symbols` option as shown in the above example.

Selecting Target Based on Target Properties

The following is an example XSDB session that demonstrates selecting a target based on target properties. It shows how to connect to the Cortex®-A9 processor of the second device when multiple devices are connected in a JTAG chain (xc7z020 and xc7z045).

```
# connect to hw_server
xsdb% conn -ho xhdbfarmrkh1
tcfchan#0
# check the jtag targets connected, the IDs listed with jtag targets are
called node IDs
xsdb% jtag targets
 1 Platform Cable USB II 0000153f74cd01
 2 arm_dap (idcode 4ba00477 irlen 4)
 3 xc7z020 (idcode 03727093 irlen 6 fpga)
 4 arm_dap (idcode 4ba00477 irlen 4)
 5 xc7z045 (idcode 03731093 irlen 6 fpga)
# check the targets connected, the IDs listed with targets are called
target IDs
xsdb% targets
 1 APU
 2 ARM Cortex-A9 MPCore #0 (Suspended)
 3 ARM Cortex-A9 MPCore #1 (Suspended)
 4 xc7z020
 5 APU
 6 ARM Cortex-A9 MPCore #0 (Running)
 7 ARM Cortex-A9 MPCore #1 (Running)
 8 xc7z045
# check jtag target properties of 2nd device (2nd ARM DAP). Note the
target_ctx here.
xsdb% jtag targets -target-properties -filter {node_id == 4}
{target_ctx jsn-DLC10-0000153f74cd01-4ba00477-1 level 1 node_id 4 is_open
1 is_active 1 is_current 1 name arm_dap jtag_cable_name {Platform
Cable USB II 0000153f74cd01} state {} jtag_cable_manufacturer Xilinx
jtag_cable_product DLC10 jtag_cable_serial 0000153f74cd01 idcode 4ba00477
irlen 4}
# using the target context, select the targets associated with the JTAG
target (2nd ARM DAP - node id = 4)
xsdb% targets -filter {jtag_device_ctx == "jsn-
DLC10-0000153f74cd01-4ba00477-1"}
 5 APU
 6 ARM Cortex-A9 MPCore #0 (Running)
 7 ARM Cortex-A9 MPCore #1 (Running)
```

Memory and Register accesses from XSCT

Memory accesses

Memory accesses in XSDB take different physical paths in the SoC devices based on the active target.

Processor targets

When a processor is selected as an active target, any memory read/write commands (mrd/mwr) run by the users are executed by the processor. The debugger injects load/store instructions into the processor to access the memory. Because the processor executes these instructions, MMU and caches come into the picture. The address of the read/write commands is treated as virtual address by the processor. Data is read from or written to caches. If MMU and caches are disabled, physical memory is accessed during load/store.

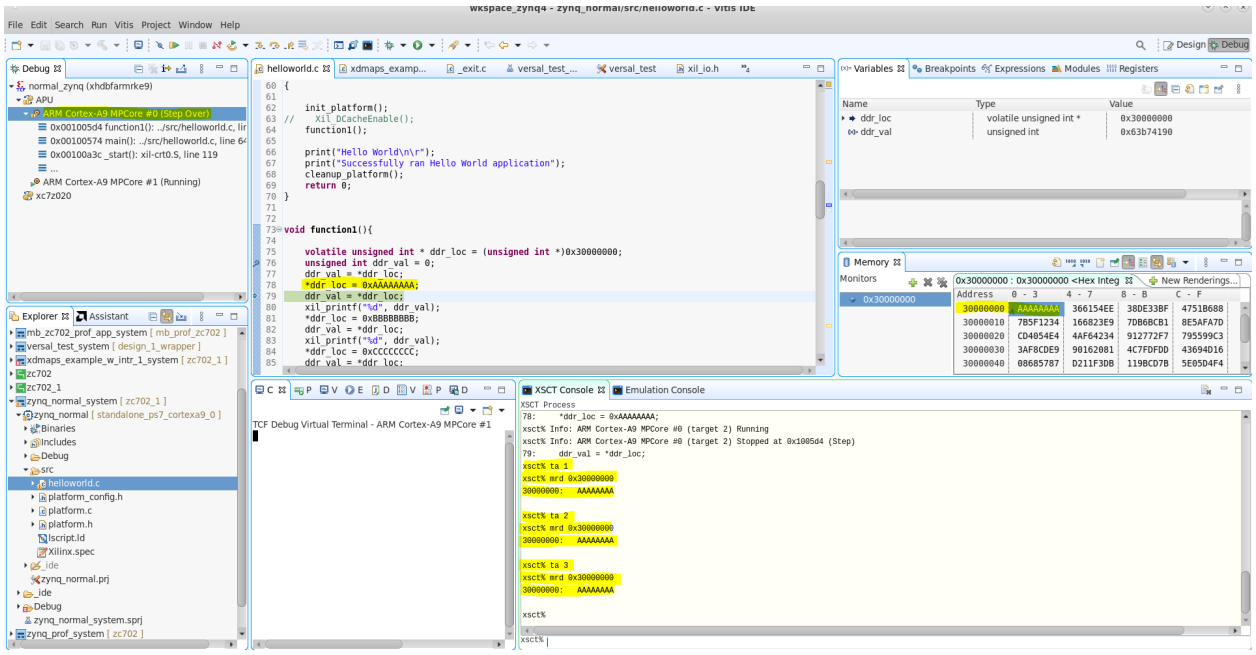
Example, processor targets are Cortex-A9, Cortex®-A53, Cortex-A72, and so on.

Non-processor Targets

When targets like APU/RPU/PSU/Versal are selected as active targets, physical memory is accessed during memory read/write commands. These targets use AXI-AP in Arm® DAP to access memory. The AXI-AP does not have access to the MMU or caches inside processor targets. However, the debugger flushes/invalidates the caches for every memory access command. Therefore, it is the same data on any target, APU/processor, even though the cache is enabled. Following are the examples.

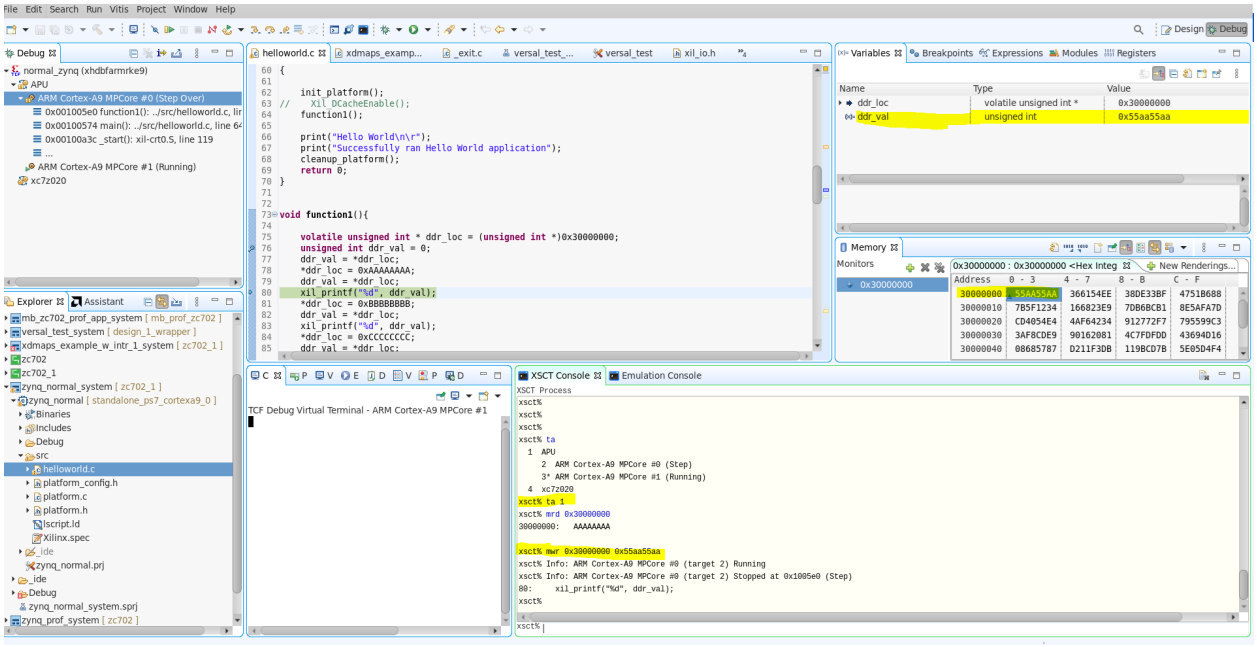
Case 1: Write 0xAAAAAAAA to the location 0x30000000 from the processor target and read it from the APU target or processor target. The data is the same.

Figure 1: **Behavior of Memory-writes from Processor Target**



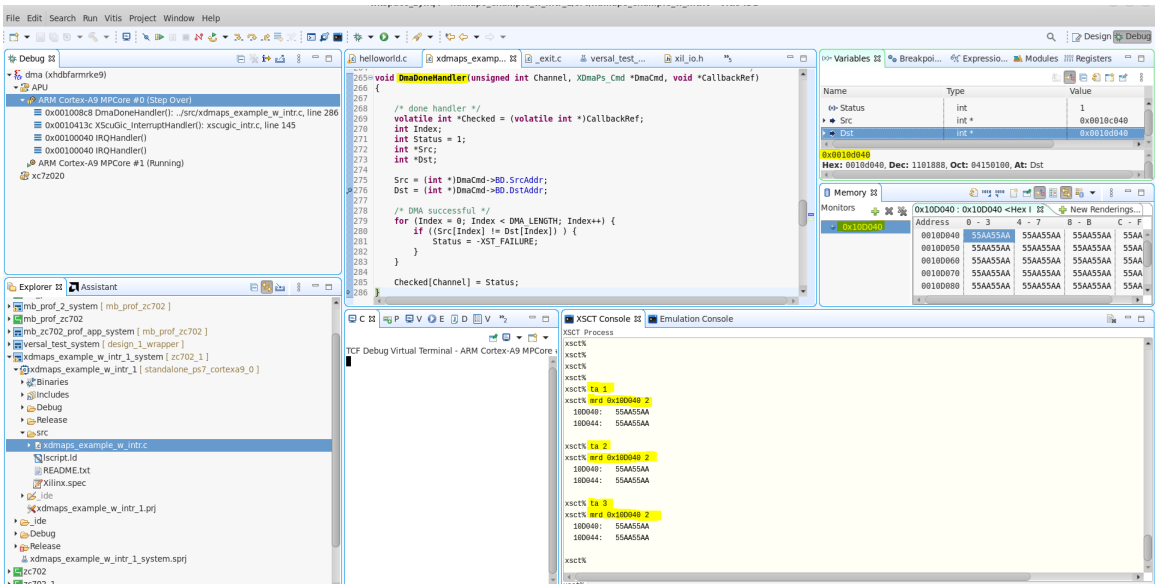
Case 2: Write 0x55AA55AA to the location 0x30000000 from the APU target and read it from the APU target or processor target. The data is the same in both targets. In the following figure, select target 1 on Zynq, which is APU, and write 0x55AA55AA. From Vitis, check the ddr_val variable from the processor target, which is updated as soon as we write from target1.

Figure 2: **Behavior of Memory-writes from APU Target**



Case 3: Perform DMA transfers from the application (processor target) by filling source location (0x10C040) with 0x55AA55AA, of size 1024 words, and destination location (0x10D040) with 0xDEADBEEF, of size 1024 words. Initiate the DMA transfer. After the transfer is done, verify the destination location from the processor and APU Targets. The data will be the same.

Figure 3: **Behavior of DMA-transfers from Processor Target**



The Effect of XPPU/XMPU on Memory Accesses

If XMPU/XPPU are configured to block accesses to a peripheral/memory through a processor or DAP, accessing such addresses thru these targets in XSDB leads to memory access errors. Memory accesses succeed only when the hardware is configured to allow access from the active target to that address. Users must use 'loadhw' or 'memmap' commands that provides access to the protected memory regions. Alternatively, '-force' option in 'mrd'/'mwr' commands can be used or set 'force-mem-access' in 'configparams' to 1.

Register Accesses

The registers list varies based on the active target. For processor targets, the registers list includes processor registers like general purpose register, PC, SP, LR, special registers, banked registers and so on.

For non-processor targets like APU/RPU/PSU/Versal, the registers list includes registers from all the peripherals in the SoC. For example, peripherals include DDRC, UART, SPI and so on.

Registers are grouped wherever applicable. Such registers can be accessed by running mrd/mwr <register-group> <register-name> ?value?

Modifying BSP Settings

Below is an example XSDB session that demonstrates building a HelloWorld application to target the MicroBlaze™ processor. The STDIN and STDOUT OS parameters are changed to use the `MDM_0`.

Note: When the BSP settings are changed, it is necessary to update the mss and regenerate the BSP sources to reflect the changes in the source file before compiling.

```
setws /tmp/wrk/workspace
app create -name mb_app -hw /tmp/wrk/kc705_system.xsa -proc microblaze_0
-os standalone -template {Hello World}
bsp config stdin mdm_0
bsp config stdout mdm_0
bsp regenerate
platform generate
app build -name mb_app

setws /tmp/wrk/workspace
app create -name mb_app -hw /tmp/wrk/kc705_system.xsa -proc microblaze_0 -
os standalone -template

{Hello World}
bsp config stdin mdm_0
bsp config stdout mdm_0
bsp regenerate
platform generate
app build -name mb_app
```

Performing Standalone Application Debug

System Command-line Tool (XSDB) can be used to debug standalone applications on one or more processor cores simultaneously. The first step involved in debugging is to connect to hw_server and select a debug target. You can now reset the system/processor core, initialize the PS if needed, program the FPGA, download an elf, set breakpoints, run the program, examine the stack trace, view local/global variables.

An example XSCT session that demonstrates standalone application debug on AMD Zynq™ 7000 SoC is as follows. Comments begin with `#`.

```
#connect to remote hw_server by specifying its url.
#If the hardware is connected to a local machine,-url option and the <url>
#are not needed. connect command returns the channel ID of the connection

xsct% connect -url TCP:xhdbfarmc7:3121 tcfchan#0

# List available targets and select a target through its id.
#The targets are assigned IDs as they are discovered on the Jtag chain,
#so the IDs can change from session to session.
#For non-interactive usage, -filter option can be used to select a target,
```

```
#instead of selecting the target through its ID

xsct% targets
  1  APU
     2  Arm Cortex-A9 MPCore #0 (Running)
     3  Arm Cortex-A9 MPCore #1 (Running)
  4  xc7z020
xsct% targets 2
# Reset the system before initializing the PS and configuring the FPGA

xsct% rst
# Info messages are displayed when the status of a core changes
Info: Arm Cortex-A9 MPCore #0 (target 2) Stopped at 0xfffffe1c (Suspended)
Info: Arm Cortex-A9 MPCore #1 (target 3) Stopped at 0xfffffe18 (Suspended)

# Configure the FPGA. When the active target is not a FPGA device,
#the first FPGA device is configured

xsct% fpga ZC702_HwPlatform/design_1_wrapper.bit
100%    3MB   1.8MB/s  00:02

# Run loadhw command to make the debugger aware of the processor cores'
memory map
xsct% loadhw ZC702_HwPlatform/system.xsa
design_1_wrapper

# Source the ps7_init.tcl script and run ps7_init and ps7_post_config
commands
xsct% source ZC702_HwPlatform/ps7_init.tcl
xsct% ps7_init
xsct% ps7_post_config

# Download the application program
xsct% dow dhrystone/Debug/dhrystone.elf
Downloading Program -- dhrystone/Debug/dhrystone.elf
     section, .text: 0x00100000 - 0x001037f3
     section, .init: 0x001037f4 - 0x0010380b
     section, .fini: 0x0010380c - 0x00103823
     section, .rodata: 0x00103824 - 0x00103e67
     section, .data: 0x00103e68 - 0x001042db
     section, .eh_frame: 0x001042dc - 0x0010434f
     section, .mmu_tbl: 0x00108000 - 0x0010bfff
     section, .init_array: 0x0010c000 - 0x0010c007
     section, .fini_array: 0x0010c008 - 0x0010c00b
     section, .bss: 0x0010c00c - 0x0010e897
     section, .heap: 0x0010e898 - 0x0010ec9f
     section, .stack: 0x0010eca0 - 0x0011149f
100%    0MB   0.3MB/s  00:00

Setting PC to Program Start Address 0x00100000

Successfully downloaded dhrystone/Debug/dhrystone.elf

# Set a breakpoint at main()
xsct% bpadd -addr &main
0

# Resume the processor core
xsct% con

# Info message is displayed when the core hits the breakpoint
xsct% Info: Arm Cortex-A9 MPCore #0 (target 2) Stopped at 0x1005a4
(Breakpoint)
```

```
# Registers can be viewed when the core is stopped
xsct% rrd
     r0: 00000000       r1: 00000000       r2: 0010e898       r3: 001042dc
     r4: 00000003       r5: 0000001e       r6: 0000ffff       r7: f8f00000
     r8: 00000000       r9: ffffffff      r10: 00000000      r11: 00000000
    r12: 0010fc90       sp: 0010fca0       lr: 001022d8       pc: 001005a4
   cpsr: 600000df      usr                fiq                irq
    abt                und                svc                mon
    vfp               cp15            Jazelle

# Memory contents can be displayed
xsct% mrd 0xe000d000
E000D000:   800A0000

# Local variables can be viewed
xsct% locals
Int_1_Loc       : 1113232
Int_2_Loc       : 30
Int_3_Loc       : 0
Ch_Index        : 0
Enum_Loc        : 0
Str_1_Loc       : char[31]
Str_2_Loc       : char[31]
Run_Index       : 1061232
Number_Of_Runs  : 2

# Local variable value can be modified
xsct% locals Number_Of_Runs 100
xsct% locals Number_Of_Runs
Number_Of_Runs  : 100

# Global variables and be displayed, and its value can be modified
xsct% print Int_Glob
Int_Glob  : 0
xsct% print -set Int_Glob 23
xsct% print Int_Glob
Int_Glob  : 23

# Expressions can be evaluated and its value can be displayed
xsct% print Int_Glob + 1 * 2
Int_Glob + 1 * 2  : 25

# Step over a line of source code
xsct% nxt
Info: Arm Cortex-A9 MPCore #0 (target 2) Stopped at 0x1005b0 (Step)

# View stack trace
xsct% bt
    0  0x1005b0 main()+12: ../src/dhry_1.c, line 91
    1  0x1022d8 _start()+88
    2  unknown-pc

# Set a breakpoint at exit and resume execution
xsct% bpadd -addr &exit
1
xsct% con
Info: Arm Cortex-A9 MPCore #0 (target 2) Running
xsct% Info: Arm Cortex-A9 MPCore #0 (target 2) Stopped at 0x103094
```

```
(Breakpoint)
xsct% bt
    0  0x103094 exit()
    1  0x1022e0 _start()+96
    2  unknown-pc
```

While a program is running on `A9 #0`, you can download another elf onto `A9 #1` and debug it, using similar steps. It is not necessary to re-connect to the hw_server, initialize the PS or configure the FPGA in such cases. You can select `A9 #1` target and download the elf and continue with further debug.

Generating SVF Files

SVF (Serial Vector Format) is an industry standard file format that is used to describe JTAG chain operations in a compact, portable fashion. An example XSDB script to generate an SVF file is as follows:

```
# Reset values of respective cores
set core 0
set apu_reset_a53 {0x380e 0x340d 0x2c0b 0x1c07}
# Generate SVF file for linking DAP to the JTAG chain
# Next 2 steps are required only for Rev2.0 silicon and above.
svf config -scan-chain {0x14738093 12 0x5ba00477 4
} -device-index 1 -linkdap -out "dapcon.svf"
svf generate
# Configure the SVF generation
svf config -scan-chain {0x14738093 12 0x5ba00477 4
} -device-index 1 -cpu-index $core -delay 10 -out "fsbl_hello.svf"
# Record writing of bootloop and release of A53 core from reset
svf mwr 0xffff0000 0x14000000
svf mwr 0xfd1a0104 [lindex $apu_reset_a53 $core]
# Record stopping the core
svf stop
# Record downloading FSBL
svf dow "fsbl.elf"
# Record executing FSBL
svf con
svf delay 100000
# Record some delay and then stopping the core
svf stop
# Record downloading the application
svf dow "hello.elf"
# Record executing application
svf con
# Generate SVF
svf generate
```

***Note*:** SVF files can only be recorded using XSDB. You can use any standard SVF player to play the SVF file.

To play a SVF file in the Vivado hardware manager, connect to a target and use the following Tcl command to play the file on the selected target.

```
execute_hw_svf <*.svf file>
```

Program U-BOOT over JTAG

```
#connecting to the target
connect -host <hostname>;

#Disable security gates to view PMU MB target
targets -set -filter {name =~ "PSU"}
 ;
#By default, JTAG security gates are enabled
#This disables security gates for DAP, PLTAP, and PMU.
mwr 0xffca0038 0x1ff
after 500
 ;
#Load and run PMU FW
targets -set -filter {name =~ "MicroBlaze PMU"}5)
dow pmufw.elf
con
after 500
 ;
#Reset A53, load and run FSBL
targets -set -filter {name =~ "Cortex-A53 #0"}
rst -processor
dow zynqmp_fsbl.elf
con
#Give FSBL time to run
after 5000
stop
 ;
#Dowloading other Softwares like U-boot ans so on, using below command
dow -data system.dtb 0x100000
dow u-boot.elf
dow bl31.elf
con
after 5000
stop
```

Running an Application in Non-Interactive Mode

System Debugger Command-line Interface (XSDB) provides a scriptable interface to run applications in non-interactive mode. To run the program in previous example using a script, create a Tcl script (and name it as, for example, `test.tcl`) with the following commands. The script can be run by passing it as a launch argument to XSDB.

```
connect -url TCP:xhdbfarmc7:3121

# Select the target whose name starts with Arm and ends with #0.
# On Zynq, this selects “Arm Cortex-A9 MPCore #0”

targets -set -filter {name =~ "Arm* #0"}
rst
fpga ZC702_HwPlatform/design_1_wrapper.bit
```

```
loadhw ZC702_HwPlatform/system.xsa
source ZC702_HwPlatform/ps7_init.tcl
ps7_init
ps7_post_config
dow dhrystone/Debug/dhrystone.elf

# Set a breakpoint at exit

bpadd -addr &exit

# Resume execution and block until the core stops (due to breakpoint)
# or a timeout of 5 sec is reached

con -block -timeout 5
```

Running Tcl Scripts

You can create Tcl scripts with XSDB commands and run them in an interactive or non-interactive mode. In the interactive mode, you can source the script at XSDB prompt. For example:

```
XSDB% source XSDB_script.tcl
```

In the non-interactive mode, you can run the script by specifying the script as a launch argument. Arguments to the script can follow the script name. For example:

```
$ XSDB XSDB_script.tcl [args]
```

The script below provides a usage example of XSDB. This script creates and builds an application, connects to a remote hw_server, initializes the Zynq PS connected to remote host, downloads and executes the application on the target. These commands can be either scripted or run interactively.

```
# Set Vitis workspace
setws /tmp/workspace
# Create application project
app create -name hello -hw /tmp/wrk/system.xsa -proc ps7_cortexa9_0 -os
standalone -lang C -template {Hello World}
app build -name hello hw_server
connect -host raptor-host
# Select a target
targets -set -nocase -filter {name =~ “Arm* #0}
# System Reset
rst -system
# PS7 initialization
namespace eval xsdb {source /tmp/workspace/hw1/ps7_init.tcl; ps7_init}
# Download the elf
dow /tmp/workspace/hello/Debug/hello.elf
# Insert a breakpoint @ main
bpadd -addr &main
# Continue execution until the target is suspended
```

```
con -block -timeout 500
# Print the target registers
puts [rrd]
# Resume the target
con
```

Switching Between XSDB and Vitis Integrated Design Environment

Below is an example XSDB session that demonstrates creating and building an application using XSDB. After execution, launch the Vitis development environment and select the workspace created using XSDB to view the updates.

***Note*:** The workspace created in XSDB can be used from Vitis IDE. However, at a time, only one instance of the tool can use the workspace.

```
# Set Vitis workspace
setws /tmp/workspace
# Create application project
app create -name hello -hw /tmp/wrk/system.xsa -proc ps7_cortexa9_0 -os
standalone -lang C -template {Hello World}
app build -name hello
```

Using JTAG UART

XSDB supports virtual UART through JTAG, which is useful when the physical UART does not exist or is non-functional. To use JTAG UART, the software application should be modified to redirect STDIO to the JTAG UART. Vitis IDE provides a CoreSight™ driver to support redirecting of STDIO to virtual UART on Arm based designs. For MB designs, the uartlite driver can be used. To use the virtual UART driver, open board support settings in Vitis IDE and can change STDIN / STDOUT to coresight/mdm.

XSDB supports virtual UART through two commands.

- `jtagterminal` - Start/Stop JTAG based Hyper Terminal. This command opens a new terminal window for STDIO. The text input from this terminal is sent to STDIN and any output from STDOUT is displayed on this terminal.
- `readjtaguart` - Start/Stop reading from JTAG UART. This command starts polling STDOUT for output and displays in on XSDB terminal or redirects it to a file.

An example XSCT session that demonstrates how to use a JTAG terminal for STDIO is as follows:

```
connect
source ps7_init.tcl
targets -set -filter {name =~"APU"}
loadhw system.xsa
stop
ps7_init
targets -set -nocase -filter {name =~ "Arm*#0"}
rst -processor
dow <app>.elf
jtagterminal
con
jtagterminal -stop #after you are done
```

An example XSCT session that demonstrates how to use the XSCT console as STDOUT for JTAG UART is as follows:

```
connect
source ps7_init.tcl
targets -set -filter {name =~"APU"}
loadhw system.xsa
stop
ps7_init
targets -set -nocase -filter {name =~ "Arm*#0"}
rst -processor
dow <app>.elf
readjtaguart
con
readjtaguart -stop #after you are done
```

An example XSCT session that demonstrates how to redirect the STDOUT from JTAG UART to a file is as follows:

```
connect
source ps7_init.tcl
targets -set -filter {name =~"APU"}
loadhw system.xsa
stop
ps7_init
targets -set -nocase -filter {name =~ "Arm*#0"}
rst -processor
dow <app>.elf
set fp [open uart.log w]
readjtaguart -handle $fp
con
readjtaguart -stop #after you are done
```

Working with Libraries

An example XSDB session that demonstrates creating a default domain and adding XILFFS and XILRSA libraries to the BSP is as follows. Create a FSBL application thereafter.

***Note*:** A normal domain/BSP does not contain any libraries.

```
setws /tmp/wrk/workspace
app create -name hello -hw /tmp/wrk/system.xsa -proc ps7_cortexa9_0 -os
standalone -lang C -template {Hello World}
bsp setlib -name xilffs
bsp setlib -name xilrsa
platform generate
app build -name hello
```

Changing the OS version.

```
bsp setosversion -ver 6.6
```

Assigning a driver to an IP.

```
bsp setdriver -ip ps7_uart_1 -driver generic -ver 2.0
```

Removing a library (removes xilrsa library from the domain/BSP).

```
bsp removelib -name xilrsa
```

Editing FSBL/PMUFW Source File

The following example shows you how to edit FSBL/PMUFW source files.

```
setws workspace
app create -name a53_app -hw zcu102 -os standalone -proc psu_cortexa53_0
#Go to “workspace/zcu102/zynqmp_fsbl” or “workspace/zcu102/zynqmp_pmufw”
and modify the source files using any editor like gedit or gvim for boot
domains zynqmp_fsbl and zynqmp_pmufw.
platform generate
```

Editing FSBL/PMUFW Settings

The following example shows you how to edit FSBL/PMUFW settings.

```
setws workspace
app create -name a53_app -hw zcu102 -os standalone -proc psu_cortexa53_0
#If you want to modify anything in zynqmp_fsbl domain use below command to
active that domain
domain active zynqmp_fsbl
#If you want to modify anything in zynqmp_pmufw domain use below command to
active that domain
domain active zynqmp_pmufw
#configure the BSP settings for boot domain like FSBL or PMUFW
bsp config -append compiler_flags -DFSBL_DEBUG_INFO
platform generate
```

Exchanging Files between Host Machine and Linux Running on QEMU

XSDB `tfile` can be used to communicate with the tcf-agent running in Linux to transfer files. To exchange file between host machine and Linux in QEMU, follow these steps:

1. Launch QEMU from the Vitis IDE by selecting **Vitis → Start/Stop Emulator**. QEMU is launched to boot Linux. The tcf-agent runs in the backend when Linux finishes booting. It is required to include the tcf-agent in the Linux root file system configuration in PetaLinux.
2. Launch XSCT and use the following commands to exchange file:
   a. Connect to the tcf-agent using XSDB:

      ```
      connect -host 127.0.0.1 -port 1440
      ```

      ***Note*:** 1440 is port forwarded by QEMU.

   b. Copy file from host to target:

      ```
      tfile copy -from-host <host_path> <target_path>
      ```

   c. Copy file from target to host:

      ```
      tfile copy -to-host <target_path> <host_path>
      ```

Loading U-Boot over JTAG

Users can load and run U-Boot over JTAG and then flash drivers in the U-Boot to program the flash.

Script to run U-Boot and download the binary file for programming to flash.

- Zynq:

```
# Connect to target
connect

# Load and run FSBL
targets -set -nocase -filter {name =~ "ARM*#0"}
dow zynq_fsbl.elf
con
after 3000
stop

# Load DTB at 0x100000
dow -data system.dtb 0x100000

# Download and run u-boot
dow uboot.elf
```

```
con
after 1000;
stop

# Download the BOOT.bin (file to program to flash) in some DDR location
which is not used for other apps.
dow -data BOOT.BIN <ddr_addr>
```

- AMD Zynq™MP:

```
# Connect to target
connect

# Disable Security gates to view PMU MB target
targets -set -nocase -filter {name =~ "*PSU*"}
mask_write 0xFFCA0038 0x1C0 0x1C0

#Load and run PMU FW
targets -set -nocase -filter {name =~ "*MicroBlaze PMU*"}
dow pmufw.elf
con
after 500

# Load and run FSBL
targets -set -nocase -filter {name =~ "*A53*#0"}
rst -proc
dow zynqmp_fsbl.elf
con
after 5000
stop

# Load DTB at 0x100000
dow -data system.dtb 0x100000

# Load and run u-boot
dow u-boot.elf
dow bl31.elf
con
after 5000
stop

# Download the BOOT.bin (file to program to flash) in some DDR location
which is not used for other apps.
dow -data BOOT.BIN <ddr_address>
```

- Versal:

```
# Connect to target
connect

# Configure the device with PDI containing PLM, necessary CDOs, u-boot,
BL31 and system.dtb
# Steps for creating this PDI (BOOT.BIN) are given in the next section
targets -set -nocase -filter {name =~ "*PMC*"}
device program BOOT.BIN

# Download the BOOT.bin (file to program to flash) in some DDR location
which is not used for other apps.
targets -set -nocase -filter {name =~ "*A72*#0"}
stop
dow -data BOOT.BIN <ddr_address>
```

Following are the steps to create BOOT.BIN used in the script:

1. Extract the PDI from the XSA - lets call it system.pdi
2. Create a BIF as shown below: bootimage.bif

```
all:
{
    image
    {
        { type = bootimage, file = system.pdi }
    }
    image
    {
        name=default_subsys, id=0x1c000000
        { load = 0x1000, file = system.dtb }
        { core = a72-0, exception_level = el-3, trustzone, file =
bl31.elf }
        { core = a72-0, exception_level = el-2, load=0x8000000, file=u-
boot.elf }
    }
}
```

3. Use Bootgen to create a new extended PDI by appending system.dtb, U-Boot and bl31 to the PDI extracted from `XSA Bootgen -arch versal -image bootimage.bif -w -o BOOT.BIN`.

Running U-Boot commands for flash programming

After loading and running U-Boot, at the U-Boot console, input the following commands:

```
sf probe 0 0 0
sf erase 0 <size of BOOT.bin>
sf write <ddr_address> 0 <size of BOOT.bin>
```

Hardware Software Interface (HSI) Commands

XSDB Interface Examples

HSI Tcl Examples

This chapter demonstrates how to load an `.xsa` file, access the hardware information, and generate BSPs, applications, and the Device Tree.

Accessing Hardware Design Data

Opening the hardware design

```
hsi::open_hw_design base_zynq_design_wrapper.xsa
base_zynq_design_imp
```

List loaded hardware designs

```
hsi::get_hw_designs
base_zynq_design_imp
```

Switch to current hardware design

```
hsi::current_hw_design
base_zynq_design_imp
```

Report properties of the current hardware design

```
common::report_property [hsi::current_hw_design]
```

Table 1: **Example Table**

Property	Type	Read Only	Visible	Value
ADDRESS_TAG	string*	true	true	base_zynq_design_i/ ps7_cortexa9_0:base_zynq_design_i base_zynq_design_i/ ps7_cortexa9_1:base_zynq_design_i
BOARD	string	true	true	xilinx.com:zc702:part0:1.1
CLASS	string	true	true	hw_design
DEVICE	string	true	true	7x020
FAMILY	string	true	true	zynq
NAME	string	true	true	base_zynq_design_imp
PACKAGE	string	true	true	clg484
PATH	string	true	true	/scratch/demo//base_zynq_design.hwh
SPEEDGRADE	string	true	true	1
SW_REPOSITORIES	string*	true	true	
TIMESTAMP	string	true	true	<current date and time>
VIVADO_VERSION	string	true	true	2014.3

List the .xsa files in the container

```
hsi::get_hw_files
base_zynq_design.hwh ps7_init.c ps7_init.h ps7_init_gpl.c
ps7_init_gpl.h ps7_init.tcl ps7_init.html
base_zynq_design_wrapper.mmi base_zynq_design_bd.tcl
```

Filter the .bit files

```
hsi::get_hw_files -filter {TYPE==bit}
base_zynq_design_wrapper.bit
```

List of external ports in the design

```
hsi::get_ports
DDR_cas_n DDR_cke DDR_ck_n DDR_ck_p DDR_cs_n DDR_reset_n
DDR_odt DDR_ras_n
DDR_we_n DDR_ba DDR_addr DDR_dm DDR_dq DDR_dqs_n DDR_dqs_p
FIXED_IO_mio
FIXED_IO_ddr_vrn FIXED_IO_ddr_vrp FIXED_IO_ps_srstb
FIXED_IO_ps_clk
FIXED_IO_ps_porb leds_4bits_tri_o
```

Reports properties of an external port

```
common::report_property [hsi::get_ports leds_4bits_tri_o]
```

Table 2: **Example Table**

Property	Type	Readonly	Visible	Value
CLASS	string	true	true	port
CLK_FREQ	string	true	true	
DIRECTION	string	true	true	0
INTERFACE	bool	true	true	0
IS_CONNECTED	bool	true	true	0
LEFT	string	true	true	3
NAME	string	true	true	leds_4bits_tri_o
RIGHT	string	true	true	0
SENSITIVITY	enum	true	true	
TYPE	enum	true	true	undef

List of IP instances in the design

```
hsi::get_cells
axi_bram_ctrl_0 axi_gpio_0 blk_mem_gen_0
processing_system7_0_axi_periph_m00_couplers_auto_pc
processing_system7_0_axi_periph_s00_couplers_auto_pc
processing_system7_0_axi_periph_xbar
rst_processing_system7_0_50M ps7_clockc_0 ps7_uart_1
ps7_pl310_0 ps7_pmu_0 ps7_qspi_0
ps7_qspi_linear_0 ps7_axi_interconnect_0 ps7_cortexa9_0
ps7_cortexa9_1 ps7_ddr_0
ps7_ethernet_0 ps7_usb_0 ps7_sd_0 ps7_i2c_0 ps7_can_0
ps7_ttc_0 ps7_gpio_0
ps7_ddrc_0 ps7_dev_cfg_0 ps7_xadc_0 ps7_ocmc_0
ps7_coresight_comp_0 ps7_gpv_0 ps7_scuc_0
ps7_globaltimer_0 ps7_intc_dist_0 ps7_l2cachec_0 ps7_dma_s
ps7_iop_bus_config_0 ps7_ram_0
ps7_ram_1 ps7_scugic_0 ps7_scutimer_0 ps7_scuwdt_0
ps7_slcr_0 ps7_dma_ns ps7_afi_0 ps7_afi_1
ps7_afi_2 ps7_afi_3 ps7_m_axi_gp0
```

#List of processors in the design

```
hsi::get_cells -filter {IP_TYPE==PROCESSOR}
ps7_cortexa9_0 ps7_cortexa9_1
```

Properties of IP instance

```
common::report_property [hsi::get_cells axi_gpio_0]
```

Table 3: **Example Table**

Property	Type	Readonly	Visible	Value
CLASS	string	true	true	cell
CONFIG.C_ALL_INPUTS	string	true	true	0
CONFIG.C_ALL_INPUTS_2	string	true	true	0

Table 3: **Example Table** *(cont'd)*

Property	Type	Readonly	Visible	Value
CONFIG.C_ALL_OUTPUTS	string	true	true	1
CONFIG.C_ALL_OUTPUTS_2	string	true	true	0
CONFIG.C_BASEADDR	string	true	true	0x41200000
CONFIG.C_DOUT_DEFAULT	string	true	true	0x00000000
CONFIG.C_DOUT_DEFAULT_2	string	true	true	0x00000000
CONFIG.C_FAMILY	string	true	true	zynq
CONFIG.C_GPIO2_WIDTH	string	true	true	32
CONFIG.C_GPIO_WIDTH	string	true	true	4
CONFIG.C_HIGHADDR	string	true	true	0x4120FFFF
CONFIG.C_INTERRUPT_PRESENT	string	true	true	0
CONFIG.C_IS_DUAL	string	true	true	0
CONFIG.C_S_AXI_ADDR_WIDTH	string	true	true	9
CONFIG.C_S_AXI_DATA_WIDTH	string	true	true	32
CONFIG.C_TRI_DEFAULT	string	true	true	0xFFFFFFFF
CONFIG.C_TRI_DEFAULT_2	string	true	true	0xFFFFFFFF
CONFIG.Component_Name	string	true	true	base_zynq_design_axi_gpio_0_0
CONFIG.EDK_IPTYPE	string	true	true	PERIPHERAL
CONFIG.GPIO2_BOARD_INTERFACE	string	true	true	Custom
CONFIG.GPIO_BOARD_INTERFACE	string	true	true	leds_4bits
CONFIG.USE_BOARD_FLOW	string	true	true	true
CONFIGURABLE	bool	true	true	0
IP_NAME	string	true	true	axi_gpio
IP_TYPE	enum	true	true	PERIPHERAL
NAME	string*	true	true	axi_gpio_0
PRODUCT_GUIDE	string	true	true	*AXI GPIO LogiCORE IP Product Guide* (PG144)
SLAVES	string	true	true	
VLNV	string	true	true	xilinx.com:ip:axi_gpio:2.0

Memory range of the Slave IPs

```
common::report_property [lindex [hsi::get_mem_ranges -of_objects
[hsi::get_cells -filter {IP_TYPE==PROCESSOR}]] 39]
```

Table 4: **Example Table**

Property	Type	Read-only	Visible	Value
BASE_NAME	string	true	true	C_BASEADDR

Table 4: **Example Table** *(cont'd)*

Property	Type	Read-only	Visible	Value
BASE_VALUE	string	true	true	0x41200000
CLASS	string	true	true	mem_range
HIGH_NAME	string	true	true	C_HIGHADDR
HIGH_VALUE	string	true	true	0x4120FFFF
INSTANCE	cell	true	true	axi_gpio_0
IS_DATA	bool	true	true	1
IS_INSTRUCTION	bool	true	true	0
MEM_TYPE	enum	true	true	REGISTER
NAME	string	true	true	axi_gpio_0

Creating Standalone Software Design and Accessing Software Information

List of the drivers in the software repository

```
hsi::get_sw_cores *uart*
uartlite_v2_01_a uartlite_v3_0 uartns550_v2_01_a
uartns550_v2_02_a uartns550_v3_0
uartns550_v3_1 uartps_v1_04_a uartps_v1_05_a uartps_v2_0
uartps_v2_1 uartps_v2_2
```

Creates software design

```
hsi::create_sw_design swdesign -proc ps7_cortexa9_0 -os standalone
swdesign
```

To switch to active software design

```
hsi::current_sw_design
swdesign
```

Properties of the current software design

```
common::report_property [hsi::current_sw_design ]
```

Table 5: **Example Table**

Property	Type	Read-only	Visible	Value
APP_COMPILER	string	FALSE	TRUE	arm-xilinx-eabi-gcc
APP_COMPILER_FLAGS	string	FALSE	TRUE	
APP_LINKER_FLAGS	string	FALSE	TRUE	
BSS_MEMORY	string	FALSE	TRUE	
CLASS	string	TRUE	TRUE	sw_design
CODE_MEMORY	string	FALSE	TRUE	
DATA_MEMORY	string	FALSE	TRUE	

Table 5: **Example Table** *(cont'd)*

Property	Type	Read-only	Visible	Value
NAME	string	TRUE	TRUE	swdesign

The drivers associated to current hardware design

```
hsi::get_drivers
axi_bram_ctrl_0 axi_gpio_0 ps7_afi_0 ps7_afi_1 ps7_afi_2
ps7_afi_3 ps7_can_0
ps7_coresight_comp_0 ps7_ddr_0 ps7_ddrc_0 ps7_dev_cfg_0
ps7_dma_ns ps7_dma_s
ps7_ethernet_0 ps7_globaltimer_0 ps7_gpio_0 ps7_gpv_0
ps7_i2c_0 ps7_intc_dist_0
ps7_iop_bus_config_0 ps7_l2cachec_0 ps7_ocmc_0 ps7_pl310_0
ps7_pmu_0 ps7_qspi_0
ps7_qspi_linear_0 ps7_ram_0 ps7_ram_1 ps7_scuc_0
ps7_scugic_0 ps7_scutimer_0
ps7_scuwdt_0 ps7_sd_0 ps7_slcr_0 ps7_ttc_0 ps7_uart_1
ps7_usb_0 ps7_xadc_0
hsi% get_osstandalone
```

Properties of the OS object

```
common::report_property[hsi::get_os]
```

Table 6: **Example Table**

Property	Type	Read-only	Visible	Value
CLASS	string	TRUE	TRUE	sw_proc
CONFIG.archiver	string	FALSE	TRUE	arm-xilinx-eabi-ar
CONFIG.compiler	string	FALSE	TRUE	arm-xilinx-eabi-gcc
CONFIG.compiler_flags	string	FALSE	TRUE	-O2 -c
CONFIG.extra_compiler_flags	string	FALSE	TRUE	-g
HW_INSTANCE	string	TRUE	TRUE	ps7_cortexa9_0
NAME	string	FALSE	TRUE	cpu_cortexa9
VERSION	string	FALSE	TRUE	2.1

Generate BSP. BSP source code will be dumped to the output directory.

```
hsi::generate_bsp -dir bsp_out
```

List of available apps in the repository

```
hsi::generate_app -lapp
peripheral_tests dhrystone empty_application hello_world
lwip_echo_server
memory_tests rsa_auth_app srec_bootloader
xilkernel_thread_demo zynq_dram_test
zynq_fsbl linux_empty_app linux_hello_world
opencv_hello_world
```

Generate template application

```
hsi::generate_app -app hello_world -proc ps7_cortexa9_0 -
dir app_out
```

Generate Device Tree. Clone device tree repo from GIT to `/device_tree_repository/device-treegenerator-master` directory.

Load the hardware design

```
hsi::open_hw_design zynq_1_wrapper.xsa
```

Cloned GIT repo path

```
hsi::set_repo_path ./device_tree_repository/device-tree-generator-master
```

Create sw design

```
hsi::create_sw_design sw1 -proc ps7_cortexa9_0 -os device_tree
```

Generate device tree

```
hsi::generate_target {dts} -dir dtg_out
```

Generating and Compiling Applications with Customized Compiler Settings and Memory Sections

#Create a software design for the template application with default compiler flags and memory section settings

```
set sw_system_1 [hsi::create_sw_design system_1 -proc microblaze_1 -os
xilkernel -app hello_world]
```

#Change compiler and its flags of the software design

```
common::set_property APP_COMPILER "mb-gcc" $sw_system_1
common::set_property -name APP_COMPILER_FLAGS -value "-DRSA_SUPPORT -
DFSBL_DEBUG_INFO"
-objects $sw_system_1
common::set_property -name APP_LINKER_FLAGS -value "-Wl,--start-group,-
lxil,-lgcc,-lc,--end-group"
-objects $sw_system_1
```

#Change memory sections

```
common::set_property CODE_MEMORY axi_bram_ctrl_1 $sw_system_1
common::set_property BSS_MEMORY axi_bram_ctrl_1 $sw_system_1
common::set_property DATA_MEMORY axi_bram_ctrl_2 $sw_system_1
```

#Genereate application for the above customized software design to Zynq_Fsbl directory

```
hsi::generate_app -dir hw_output -compile
```

Generating and Compiling BSP with Advanced Driver/Library/OS/Processor Configuration

#Create a software design for the template application with default compiler flags and memory section settings

```
set sw_system_1 [hsi::create_sw_design system_1 -proc microblaze_1 -os
xilkernel ]
```

#Get the old driver object

```
set old_driver [hsi::get_drivers myip1]
```

#Set repository path to find the custom drivers and libraries

```
hsi::set_repo_path ./my_local_sw_repository
```

#Set the new driver name and version to old driver object

```
common::set_property NAME myip1_custom_driver $old_driver
common::set_property VERSION 1.0 $old_driver
```

#Change default OS configuration to desired one

```
set OS [hsi::get_os]
common::set_property CONFIG.systmr_dev axi_timer_0 $OS
common::set_property CONFIG.stdin axi_uartlite_0 $OS
common::set_property CONFIG.stdout axi_uartlite_0 $OS
```

#Add custom library to software design

```
hsi::add_library xilflash
```

#Get all the properties of the library, only read_only = false properties can be changed

```
common::report_property [hsi::get_libs xilflash]
```

#Change the default configuration of the library

```
set lib [hsi::get_libs xilflash]
common::set_property CONFIG.enable_amd true $lib
common::set_property CONFIG.enable_intel false $lib
```

#Generate the BSP with the above configuration

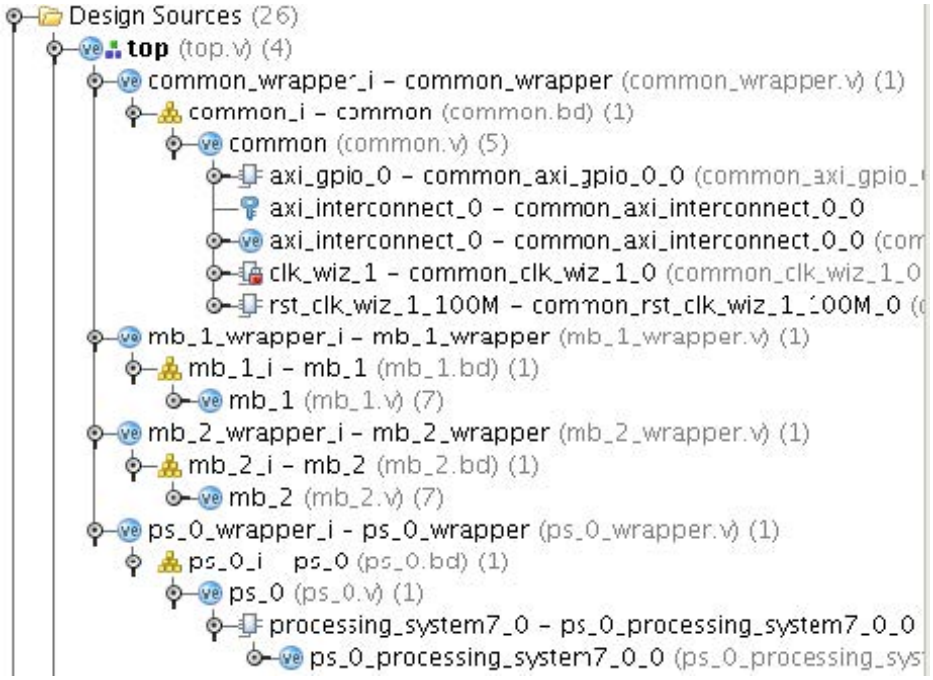
```
hsi::generate_bsp -dir advanced_bsp -compile
```

#Delete the library added to software design

```
hsi::delete_objs $lib
```

Generating and Compiling BSP for a Multi-Block Design

Figure 4: **Example Design with Multiple Block Design Instances in the Active Top Design**



#Open hardware design with multiple block design instances

```
hsi% hsi::open_hw_design system_wrapper.xsa
design_1_wrapper
```

#Get the hardware cell instances

***Note*:** Cell instances from all the block designs in the top are shown andtheir names are prefixed with their hierarchy

```
hsi% join [get_cells ] \n
ps_0_wrapper_i_ps_0_i_processing_system7_0
ps7_uart_1
ps7_qspi_0
ps7_cortexa9_0
ps7_cortexa9_1
ps7_ddr_0
```

```
ps7_ethernet_0
....
mb_2_wrapper_i_mb_2_i_axi_gpio_0
mb_2_wrapper_i_mb_2_i_mdm_1
mb_2_wrapper_i_mb_2_i_microblaze_0
mb_2_wrapper_i_mb_2_i_microblaze_0_axi_periph
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_dlmb_bram_if_cntlr
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_dlmb_v10
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_ilmb_bram_if_cntlr
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_ilmb_v10
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_lmb_bram
mb_2_wrapper_i_mb_2_i_rst_clk_wiz_1_100M
mb_1_wrapper_i_mb_1_i_axi_gpio_0
mb_1_wrapper_i_mb_1_i_mdm_1
mb_1_wrapper_i_mb_1_i_microblaze_0
mb_1_wrapper_i_mb_1_i_microblaze_0_axi_periph
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_dlmb_bram_if_cntlr
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_dlmb_v10
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_ilmb_bram_if_cntlr
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_ilmb_v10
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_lmb_bram
mb_1_wrapper_i_mb_1_i_rst_clk_wiz_1_100M
common_wrapper_i_common_i_axi_gpio_0
common_wrapper_i_common_i_axi_interconnect_0
common_wrapper_i_common_i_clk_wiz_1
common_wrapper_i_common_i_rst_clk_wiz_1_100M
```

#Generate BSP for a processor in bsp_out directory and compile the bsp sources

```
hsi::generate_bsp -proc mb_2_wrapper_i_mb_2_i_microblaze_0 -dir bsp_out -
compile
ls ./bsp_out/mb_2_wrapper_i_mb_2_i_microblaze_0
code
indent
lib
libsrc
```

HSI Input and Output Files and Specifications

Input Files

XSA

AMD Support Archive (`.xsa`) is an AMD proprietary file format and only AMD software tools understand it. Third-party software tools can communicate to the XSDB Tcl interface to extract data from the `.xsa` file.

Note: AMD does not recommend manually editing the XSA file or altering its contents.

XSA is a container and contains:

- One or more `.hwh` files
  - AMD Vivado™ tool version, part, and board tag information
  - IP - instance, name, VLNV, and parameters

    - Memory Map information of the processors
    - Internal Connectivity information (including interrupts, clocks, etc.) and external ports information
- BMM/MMI and BIT files
- User and HLS driver files
- Other meta-data files

Software Repository

Default Repositories

By default, the tool scans the following repositories for software components:

- `<install>/data/embeddedsw/lib/XilinxProcessorIPLib`
- `<install>/data/embeddedsw/lib`
- `<install>/data/embeddedsw/ThirdParty`

GIT Repositories

The Device Tree repository can be cloned from Xilinx GIT. Use the `set_repo_path` Tcl command to specify the cloned GIT repository.

User Repositories

You can create drivers, BSPs, and Apps in an example directory structure format, as illustrated in the figure above. Use the `set_repo_path` Tcl command to specify the user repository.

Search Priority Mechanism

The tool uses a search priority mechanism to locate drivers and libraries, as follows:

1. Search the repositories under the library path directory specified using the `set_repo_path` Tcl command.
2. Search the default repositories described above.

Output Files

The tool generates directories, files, and the software design file (MSS) in the `<your_project>` directory. For every processor instance in the MSS file, the tool generates a directory with the name of the processor instance. Within each processor instance directory the tool generates the following directories and files.

- **The include Directory:** The include directory contains C header files needed by drivers. The include file xparameters.h is also created using the tool in this directory. This file defines base addresses of the peripherals in the system, #defines needed by drivers, OSs, libraries, and user programs, as well as function prototypes.
  - The Microprocessor Driver Definition (MDD) file for each driver specifies the definitions that must be customized for each peripheral that uses the driver. See Microprocessor Library Definition Overview.
  - The Microprocessor Library Definition (MLD) file for each OS and library specifies the definitions that you must customize. See Microprocessor Library Definition Overview.
- **The lib Directory:** The lib directory contains `libc.a`, `libm.a`, and `libxil.a` libraries. The libxil library contains driver functions that the particular processor can access. For more information about the libraries, refer to the introductory section of the *BSP and Libraries Document Collection* (UG643).
- **The libsrc Directory:** The libsrc directory contains intermediate files and make files needed to compile the OSs, libraries, and drivers. The directory contains peripheral-specific driver files, BSP files for the OS, and library files that are copied from install, as well as your driver, OS, and library directories.
- **The code Directory:** The code directory is a repository for tool executables. The tool creates an `xmdstub.elf` file (for the MicroBlaze™ processor on-board debug) in this directory.

***Note*:** The tool removes these directories every time you run the it. You must put your sources, executables, and any other files in an area that you create.

Generating Libraries and Drivers

This section provides an overview of generating libraries and drivers. The hardware specification file and the MSS files define a system. For each processor in the system, the tool finds the list of addressable peripherals. For each processor, a unique list of drivers and libraries are built. The tool does the following for each processor:

- Builds the directory structure, as defined in Output Files.
- Copies the necessary source files for the drivers, OSs, and libraries into the processor instance specific area: OUTPUT_DIR/processor_instance_name/libsrc.
- Calls the Design Rule Check (DRC) procedure, which is defined as an option in the MDD or MLD file, for each of the drivers, OSs, and libraries visible to the processor.
- Calls the generate Tcl procedure (if defined in the Tcl file associated with an MDD or MLD file) for each of the drivers, OSs, and libraries visible to the processor. This generates the necessary configuration files for each of the drivers, OSs, and libraries in the include directory of the processor.
- Calls the `post_generate` Tcl procedure (if defined in the Tcl file associated with an MDD or MLD file) for each of the drivers, OSs, and libraries visible to the processor.

- Runs make (with targets include and libs) for the OSs, drivers, and libraries specific to the processor. On the Linux platform, the gmake utility is used, while on NT platforms, make is used for compilation.
- Calls the `execs_generate` Tcl procedure (if defined in the Tcl file associated with an MDD or MLD file) for each of the drivers, OSs, and libraries visible to the processor.

MDD, MLD, and Tcl

A driver or library has two associated data files:

- **Data Definition File (MDD or MLD file):** This file defines the configurable parameters for the driver, OS, or library.
- **Data Generation File (Tcl):** This file uses the parameters configured in the MSS file for a driver, OS, or library to generate data. Data generated includes but is not limited to generation of header files, C files, running DRCs for the driver, OS, or library, and generating executables.

  The Tcl file includes procedures that tool calls at various stages of its execution. Various procedures in a Tcl file include:

  - **DRC:** The name of DRC given in the MDD or MLD file.
  - **generate:** A tool-defined procedure that is called after files are copied.
  - **post_generate:** A tool-defined procedure that is called after generate has been called on all drivers, OSs, and libraries.
  - **execs_generate:** A tool-defined procedure that is called after the BSPs, libraries, and drivers have been generated.

***Note*:** The data generation (Tcl) file is not necessary for a driver, OS, or library.

For more information about the Tcl procedures and MDD/MLD related parameters, refer to Microprocessor Driver Definition (MDD) and Microprocessor Library Definition (MLD).

MSS Parameters

For a complete description of the MSS format and all the parameters that MSS supports, refer to MSS Overview.

Drivers

Most peripherals require software drivers. The peripherals are shipped with associated drivers, libraries and BSPs. Refer to the Device Driver Programmer Guide for more information on driver functions. This guide can be found in the `<install_directory>\vitis \<version>\data\embeddedsw\doc`.

The MSS file includes a driver block for each peripheral instance. The block contains a reference to the driver by name (DRIVER_NAME parameter) and the driver version (DRIVER_VER). There is no default value for these parameters.

A driver has an associated MDD file and a Tcl file.

- The driver MDD file is the data definition file and specifies all configurable parameters for the drivers.
- Each MDD file has a corresponding Tcl file which generates data that includes generation of header files, generation of C files, running DRCs for the driver, and generating executables.

You can write your own drivers. These drivers must be in a specific directory under / or / drivers, as shown in the figure in Software Repository.

- The DRIVER_NAME attribute allows you to specify any name for your drivers, which is also the name of the driver directory.
- The source files and make file for the driver must be in the `/src` subdirectory under the / directory.
- The make file must have the targets `/include` and `/libs`.
- Each driver must also contain an MDD file and a Tcl file in the `/data` subdirectory.

Open the existing driver files to get an understanding of the required structure.

Refer to Microprocessor Driver Definition (MDD) for details on how to write an MDD and its corresponding Tcl file.

Libraries

The MSS file includes a library block for each library. The library block contains a reference to the library name (LIBRARY_NAME parameter) and the library version (LIBRARY_VER). There is no default value for these parameters. Each library is associated with a processor instance specified using the PROCESSOR_INSTANCE parameter. The library directory contains C source and header files and a make file for the library.

The MLD file for each library specifies all configurable options for the libraries and each MLD file has a corresponding Tcl file.

You can write your own libraries. These libraries must be in a specific directory under `/sw_services` as shown in the figure in Software Repository.

- The LIBRARY_NAME attribute lets you specify any name for your libraries, which is also the name of the library directory.
- The source files and make file for the library must be in the `/src` subdirectory under the / directory.
- The make file must have the targets `/include` and `/libs`.

- Each library must also contain an MLD file and a Tcl file in the /data subdirectory.

Refer to the existing libraries for more information about the structure of the libraries.

Refer to Microprocessor Library Definition (MLD) for details on how to write an MLD and its corresponding Tcl file.

OS Block

The MSS file includes an OS block for each processor instance. The OS block contains a reference to the OS name (OS_NAME parameter), and the OS version (OS_VER). There is no default value for these parameters. The BSP directory contains C source and header files and a make file for the OS.

The MLD file for each OS specifies all configurable options for the OS. Each MLD file has a corresponding Tcl file associated with it. Refer to Microprocessor Library Definition (MLD) and Microprocessor Software Specification (MSS).

You can write your own OSs. These OSs must be in a specific directory under `/bsp`, as shown in the figure in Software Repository.

- The OS_NAME attribute allows you to specify any name for your OS, which is also the name of the OS directory.
- The source files and make file for the OS must be in the src subdirectory under the / directory.
- The make file should have the targets `/include` and `/libs`.
- Each OS must contain an MLD file and a Tcl file in the `/data` subdirectory.

Look at the existing OSs to understand the structures. See Microprocessor Library Definition (MLD) Overview for details on how to write an MLD and its corresponding Tcl file, refer to the Device Driver Programmer Guide. This guide is located in your Vitis software platform installation in `<install_directory>\vitis\<version> \data\embeddedsw\doc`.

Microprocessor Software Specification (MSS)

MSS Overview

The MSS file contains directives for customizing operating systems (OSs), libraries, and drivers.

MSS Format

An MSS file is case insensitive and any reference to a file name or instance name in the MSS file is also case sensitive. Comments can be specified anywhere in the file. A pound (#) character denotes the beginning of a comment, and all characters after it, right up to the end of the line, are ignored. All white spaces are also ignored and carriage returns act as sentence delimiters.

The keywords that are used in an MSS file are as follows:

- **BEGIN:** The keyword begins a driver, processor, or file system definition block. BEGIN should be followed by the driver, processor, or filesys keywords.
- **END:** This keyword signifies the end of a definition block.
- **PARAMETER:** The MSS file has a simple name = value format for statements. The PARAMETER keyword is required before NAME and VALUE pairs. The format for assigning a value to a parameter is parameter name = value. If the parameter is within a BEGIN-END block, it is a local assignment; otherwise it is a global (system level) assignment.

Requirements:

The syntax of various files that the embedded development tools use is described by the Platform Specification Format (PSF). The current PSF version is 2.1.0. The MSS file should also contain version information in the form of parameter Version = 2.1.0, which represents the PSF version 2.1.0.

MSS Example:

An example MSS file follows:

```
parameter VERSION = 2.1.0
BEGIN OS
parameter PROC_INSTANCE = my_microblaze
parameter OS_NAME = standalone
parameter OS_VER = 1.0
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
END
BEGIN PROCESSOR
parameter HW_INSTANCE = my_microblaze
parameter DRIVER_NAME = cpu
parameter DRIVER_VER = 1.0
parameter XMDSTUB_PERIPHERAL = my_jtag
END
BEGIN DRIVER
parameter HW_INSTANCE = my_intc
parameter DRIVER_NAME = intc
parameter DRIVER_VER = 1.0
END
BEGIN DRIVER
parameter HW_INSTANCE = my_uartlite_1
parameter DRIVER_VER = 1.0
parameter DRIVER_NAME = uartlite
END
BEGIN DRIVER
```

```
parameter HW_INSTANCE = my_uartlite_2
parameter DRIVER_VER = 1.0
parameter DRIVER_NAME = uartlite
END
BEGIN DRIVER
parameter HW_INSTANCE = my_timebase_wdt
parameter DRIVER_VER = 1.0
parameter DRIVER_NAME = timebase_wdt
END
BEGIN LIBRARY
parameter LIBRARY_NAME = XilMfs
parameter LIBRARY_VER = 1.0
parameter NUMBYTES = 100000
parameter BASE_ADDRESS = 0x80f00000
END
```

Global Parameters

These parameters are system-specific parameters and do not relate to a particular driver, file system, or library.

PSF Version

This option specifies the PSF version of the MSS file. This option is mandatory, and is formatted as:

```
parameter VERSION = 2.1.0
```

Instance-Specific Parameters

OS, Driver, Library, and Processor Block Parameters

The following list shows the parameters that can be used in OS, driver, library, and processor blocks.

PROC_INSTANCE

This option is required for the OS associated with a processor instances specified in the hardware database, and is formatted as:

```
parameter PROC_INSTANCE = <instance_name>
```

All operating systems require processor instances to be associated with them. The instance name that is given must match the name specified in the hardware database.

HW_INSTANCE

This option is required for drivers associated with peripheral instances specified in the hardware database and is formatted as:

```
parameter HW_INSTANCE = <instance_name>
```

All drivers in software require instances to be associated with the drivers. Even a processor definition block should refer to the processor instance. The instance name that is given must match the name specified in the BD file.

OS_NAME

This option is needed for processor instances that have OSs associated with them and is formatted as:

```
parameter OS_NAME = standalone
```

OS_VER

The OS version is set using the OSVER option and is formatted as:

```
parameter OS_VER = 1.0
```

This version is specified as x.y, where x and y are digits. This is translated to the OS directory searched as follows:

```
OS_NAME_vx_y
```

The MLD (Microprocessor Library Definition) files needed for each OS should be named OS_NAME.mld and should be present in a subdirectory data/ within the driver directory. Refer to Microprocessor Library Definition (MLD) for more information.

DRIVER_NAME

This option is needed for peripherals that have drivers associated with them and is formatted as:

```
parameter DRIVER_NAME = uartlite
```

Library Generator copies the driver directory specified to the `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the drivers using makefiles provided.

DRIVER_VER

The driver version is set using the DRIVER_VER option, and is formatted as:

```
parameter DRIVER_VER = 1.0
```

This version is specified as x.y, where x and y are digits. This is translated to the driver directory searched as follows:

```
DRIVER_NAME_vx_y
```

The MDD (Microprocessor Driver Definition) files needed for each driver should be named `DRIVER_NAME_v2_1_0.mdd` and should be present in a subdirectory `data/` within the driver directory. Refer to Microprocessor Driver Definition (MDD) for more information.

LIBRARY_NAME

This option is needed for libraries, and is formatted as:

```
parameter LIBRARY_NAME = xilmfs
```

The tool copies the library directory specified in the `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the libraries using makefiles provided.

LIBRARY_VER

The library version is set using the LIBRARY_VER option and is formatted as:

```
parameter LIBRARY_VER = 1.0
```

This version is specified as x.y, where x and y are digits. This is translated to the library directory searched by the tool as follows:

```
LIBRARY_NAME_vx_y
```

The MLD (Microprocessor Library Definition) files needed for each library should be named `LIBRARY_NAME.mld` and should be present in a subdirectory `data/` within the library directory. Refer to Microprocessor Library Definition (MLD) for more information.

MLD/MDD Specific Parameters

Parameters specified in the MDD/MLD file can be overwritten in the MSS file and formatted as:

```
parameter PARAM_NAME = PARAM_VALUE
```

See Microprocessor Library Definition (MLD)) and Microprocessor Driver Definition (MDD) for more information.

OS-Specific Parameters

The following list identifies all the parameters that can be specified only in an OS definition block.

STDIN

Identify the standard input device with the STDIN option, which is formatted as:

```
parameter STDIN = instance_name
```

STDOUT

Identify the standard output device with the STDOUT option, which is formatted as:

```
parameter STDOUT = instance_name
```

Example: MSS Snippet Showing OS options

```
BEGIN OS
parameter PROC_INSTANCE = my_microblaze
parameter OS_NAME = standalone
parameter OS_VER = 1.0
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
END
```

Processor-Specific Parameters

Following is a list of all of the parameters that can be specified only in a processor definition block.

XMDSTUB_PERIPHERAL

The peripheral that is used to handle the XMDStub should be specified in the XMDSTUB_PERIPHERAL option. This is useful for the MicroBlaze™ processor only, and is formatted as follows:

```
parameter XMDSTUB_PERIPHERAL = instance_name
```

COMPILER

This option specifies the compiler used for compiling drivers and libraries. The compiler defaults to `mb-gcc` or `powerpc-eabi-gcc` depending on whether the drivers are part of the MicroBlaze processor or PowerPC processor instance. Any other compatible compiler can be specified as an option, and should be formatted as follows:

This example denotes the Diab compiler as the compiler to be used for drivers and libraries.

ARCHIVER

This option specifies the utility to be used for archiving object files into libraries. The archiver defaults to `mb-ar` or `powerpc-eabi-ar` depending on whether or not the drivers are part of the MicroBlaze or PowerPC processor instance. Any other compatible archiver can be specified as an option, and should be formatted as follows:

```
parameter ARCHIVER = ar
parameter COMPILER = dcc
```

This example denotes the archiver ar to be used for drivers and libraries.

COMPILER_FLAGS

This option specifies compiler flags to be used for compiling drivers and libraries. If the option is not specified, the tool automatically uses platform and processor-specific options. This option should not be specified in the MSS file if the standard compilers and archivers are used.

The COMPILER_FLAGS option can be defined in the MSS if there is a need for custom compiler flags that override generated flags. The EXTRA_COMPILER_FLAGS option is recommended if compiler flags must be appended to the ones already generated.

Format this option as follows:

```
parameter COMPILER_FLAGS = “ “
```

EXTRA_COMPILER_FLAGS

This option can be used whenever custom compiler flags need to be used in addition to the automatically generated compiler flags, and should be formatted as follows:

```
parameter EXTRA_COMPILER_FLAGS = -g
```

This example specifies that the drivers and libraries must be compiled with debugging symbols in addition to the generated COMPILER_FLAGS.

Example: MSS Snippet Showing Processor Options

```
BEGIN PROCESSOR
parameter HW_INSTANCE = my_microblaze
parameter DRIVER_NAME = cpu
parameter DRIVER_VER = 1.00.a
parameter DEFAULT_INIT = xmdstub
parameter XMDSTUB_PERIPHERAL = my_jtag
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
parameter COMPILER = mb-gcc
parameter ARCHIVER = mb-ar
parameter EXTRA_COMPILER_FLAGS = -g -O0
parameter OS = standalone
END
```

Microprocessor Library Definition (MLD)

Microprocessor Library Definition Overview

This section describes the Microprocessor Library Definition (MLD) format, Platform Specification Format 2.1.0. An MLD file contains directives for customizing software libraries and generating Board Support Packages (BSP) for Operating Systems (OS). This document describes the MLD format and the parameters that can be used to customize libraries and OSs.

Requirements

Each OS and library has an MLD file and a Tcl (Tool Command Language) file associated with it. The MLD file is used by the Tcl file to customize the OS or library, depending on different options in the MSS file. For more information on the MSS file format, see Microprocessor Software Specification (MSS). The OS and library source files and the MLD file for each OS and library must be located at specific directories to find the files and libraries.

MLD Library Definition Files

Library Definition involves defining Data Definition (MLD) and a Data Generation (Tcl) files.

Data Definition File

The MLD file (named as `<library_name>.mld` or `<os_name>.mld`) contains the configurable parameters. A detailed description of the various parameters and the MLD format is described in MLD Parameter Descriptions.

Data Generation File

The second file (named as `<library_name>.tcl` or `<os_name>.tcl`, with the filename being the same as the MLD filename) uses the parameters configured in the MSS file for the OS or library to generate data. Data generated includes, but is not limited to, header files, C files, DRCs for the OS or library, and executables. The Tcl file includes procedures that are called by the tool at various stages of its execution. Various procedures in a Tcl file include the following:

- DRC (the name of the DRC given in the MLD file)
- `generate` (tool defined procedure) called after OS and library files are copied
- `post_generate` (tool defined procedure) called after generate has been called on all OSs, drivers, and libraries
- `execs_generate` (a tool-defined procedure) called after the BSPs, libraries, and drivers have been generated

***Note*:** An OS/library does not require a data generation file (Tcl file).

MLD Format Specification

The MLD format specification involves the MLD file format specification and the Tcl file format specification. The following subsections describe the MLD.

MLD File Format Specification

The MLD file format specification involves the description of configurable parameters in an OS or a library. The format used to describe this section is discussed in MLD Parameter Descriptions.

Tcl File Format Specification

Each OS and library has a Tcl file associated with the MLD file. This Tcl file has the following sections:

- **DRC:** Contains Tcl routines that validate your OS and library parameters for consistency.
- **Generation:** Contains Tcl routines that generate the configuration header and C files based on the library parameters.

MLD Design Rule Check Section

```
proc mydrc { handle } { }
```

The DRC function could be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (which the tool builds using the hardware (XSA) and software (MSS) database files) to read the parameter values that you set. The handle is associated with the current library in the database. The DRC procedure can get the OS and library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameters.

For errors, DRC procedures call the Tcl error command error "error msg" that displays in an error page.

For warnings, DRC procedures return a string value that can be printed on the console.

On success, DRC procedures return without any value.

MLD Format Examples

This section explains the MLD format through an example MLD file and its corresponding Tcl file.

Example: MLD File for a Library

Following is an example of an MLD file for the xilmfs library.

```
option psf_version = 2.1.0 ;
```

`option` is a keyword identified by the tool. The option name following the `option` keyword is a directive to the tool to do a specific action.

The `psf_version` of the MLD file is defined to be 2.1 in this example. This is the only option that can occur before a BEGIN LIBRARY construct now.

```
BEGIN LIBRARY xilmfs
```

The `BEGIN LIBRARY` construct defines the start of a library named `xilmfs`.

```
option DESC = "Xilinx Memory File System" ;
option drc = mfs_drc ;
option copyfiles = all;
option REQUIRES_OS = (standalone xilkernel freertos_zynq);
option VERSION = 2.0;
option NAME = xilmfs;
```

The `NAME` option indicates the name of the driver. The `VERSION` option indicates the version of the driver.

The `COPYFILES` option indicates the files to be copied for the library. The DRC option specifies the name of the Tcl procedure that the tool invokes while processing this library. The `mfs_drc` is the Tcl procedure in the `xilmfs.tcl` file that would be invoked while processing the xilmfs library.

```
PARAM name = numbytes, desc = "Number of Bytes", type = int, default =
100000, drc = drc_numbytes ;
PARAM name = base_address, desc = "Base Address", type = int, default =
0x10000, drc = drc_base_address ;
PARAM name = init_type, desc = "Init Type", type = enum, values = ("New
file system"=MFSINIT_NEW,
"MFS Image"=MFSINIT_IMAGE, "ROM Image"=MFSINIT_ROM_IMAGE), default =
MFSINIT_NEW ;
PARAM name = need_utils, desc = "Need additional Utilities?", type =
bool, default = false ;
```

`PARAM` defines a library parameter that can be configured. Each `PARAM` has the following properties associated with it, whose meanings are self-explanatory: `NAME`, `DESC`, `TYPE`, `DEFAULT`, `RANGE`, and `DRC`. The property `VALUES` defines the list of possible values associated with an `ENUM` type.

```
BEGIN INTERFACE file
PROPERTY HEADER="xilmfs.h" ;
FUNCTION NAME=open, VALUE=mfs_file_open ;
FUNCTION NAME=close, VALUE=mfs_file_close ;
FUNCTION NAME=read, VALUE=mfs_file_read ;
FUNCTION NAME=write, VALUE=mfs_file_write ;
FUNCTION NAME=lseek, VALUE=mfs_file_lseek ;
END INTERFACE
```

An interface contains a list of standard functions. A library defining an interface should have values for the list of standard functions. It must also specify a header file where all the function prototypes are defined.

`PROPERTY` defines the properties associated with the construct defined in the `BEGIN` construct. Here `HEADER` is a property with value `xilmfs.h`, defined by the file interface. `FUNCTION` defines a function supported by the interface.

The `open`, `close`, `read`, `write`, and `lseek` functions of the file interface have the values `mfs_file_open`, `mfs_file_close`, `mfs_file_read`, `mfs_file_write`, and `mfs_file_lseek`. These functions are defined in the header file `xilmfs.h`.

```
BEGIN INTERFACE filesystem
```

`BEGIN INTERFACE` defines an interface the library supports. Here, `file` is the name of the interface.

```
PROPERTY HEADER="xilmfs.h" ;
FUNCTION NAME=cd, VALUE=mfs_change_dir ;
FUNCTION NAME=opendir, VALUE=mfs_dir_open ;
FUNCTION NAME=closedir, VALUE=mfs_dir_close ;
FUNCTION NAME=readdir, VALUE=mfs_dir_read ;
FUNCTION NAME=deletedir, VALUE=mfs_delete_dir ;
FUNCTION NAME=pwd, VALUE=mfs_get_current_dir_name ;
FUNCTION NAME=rename, VALUE=mfs_rename_file ;
FUNCTION NAME=exists, VALUE=mfs_exists_file ;
FUNCTION NAME=delete, VALUE=mfs_delete_file ;
END INTERFACE
END LIBRARY
```

`END` is used with the construct name that was used in the `BEGIN` statement. Here, `END` is used with `INTERFACE` and `LIBRARY` constructs to indicate the end of each of `INTERFACE` and `LIBRARY` constructs.

Example: Tcl File of a Library

The following is the `xilmfs.tcl` file corresponding the `xilmfs.mld` file described in the previous section. The `mfs_drc` procedure would be invoked for the `xilmfs` library while running DRCs for libraries. The generate routine generates constants in a header file and a c file for the `xilmfs` library based on the library definition segment in the MSS file.

```
proc mfs_drc {lib_handle} {
puts "MFS DRC ..."
}
proc mfs_open_include_file {file_name} {
set filename [file join "../../include/" $file_name]
if {[file exists $filename]} {
set config_inc [open $filename a]
} else {
set config_inc [open $filename a]
::hsi::utils::write_c_header $config_inc "MFS Parameters"
}
return $config_inc
```

```
}
proc generate {lib_handle} {
puts "MFS generate ..."
file copy "src/xilmfs.h" "../../include/xilmfs.h"
set conffile [mfs_open_include_file "mfs_config.h"]
puts $conffile "#ifndef _MFS_CONFIG_H"
puts $conffile "#define _MFS_CONFIG_H"
set need_utils [common::get_property CONFIG.need_utils $lib_handle]
if {$need_utils} {
# tell libgen or xps that the hardware platform needs to provide
stdio functions
# inbyte and outbyte to support utils
puts $conffile "#include <stdio.h>"
}
puts $conffile "#include <xilmfs.h>"
set value [common::get_property CONFIG.numbytes $lib_handle]
puts $conffile "#define MFS_NUMBYTES $value"
set value [common::get_property CONFIG.base_address $lib_handle]
puts $conffile "#define MFS_BASE_ADDRESS $value"
set value [common::get_property CONFIG.init_type $lib_handle]
puts $conffile "#define MFS_INIT_TYPE $value"
puts $conffile "#endif"
close $conffile
}
```

Example: MLD File for an OS

An example of an MLD file for the standalone OS is given below:

```
option psf_version = 2.1.0 ;
```

`option` is a keyword identified by the tool. The option name following the `option` keyword is a directive to the tool to do a specific action. Here the psf_version of the MLD file is defined to be 2.1. This is the only option that can occur before a `BEGIN` OS construct at this time.

```
BEGIN OS standalone
```

The `BEGIN` OS construct defines the start of an OS named standalone.

```
option DESC = "Generate standalone BSP";
option COPYFILES = all;
```

The `DESC` option gives a description of the MLD. The `COPYFILES` option indicates the files to be copied for the OS.

```
PARAM NAME = stdin, DESC = "stdin peripheral ", TYPE =
peripheral_instance, REQUIRES_INTERFACE = stdin, DEFAULT = none; PARAM
NAME = stdout, DESC = "stdout peripheral ", TYPE = peripheral_instance,
REQUIRES_INTERFACE = stdout, DEFAULT = none ; PARAM NAME = need_xilmalloc,
DESC = "Need xil_malloc?", TYPE = bool, DEFAULT = false ;
```

PARAM defines an OS parameter that can be configured. Each PARAM has the following, associated properties: NAME, DESC, TYPE, DEFAULT, RANGE, DRC. The property VALUES defines the list of possible values associated with an ENUM type.

```
END OS
```

END is used with the construct name that was used in the BEGIN statement. Here END is used with OS to indicate the end of OS construct.

Example: Tcl File of an OS

The following is the standalone.tcl file corresponding to the standalone.mld file described in the previous section. The generate routine generates constants in a header file and a c file for the xilmfs library based on the library definition segment in the MSS file.

```
proc generate {os_handle} {
    global env
    set need_config_file "false"
    # Copy over the right set of files as src based on processor type
    set sw_proc_handle [get_sw_processor]
    set hw_proc_handle [get_cells [get_property HW_INSTANCE
$sw_proc_handle] ]
    set proctype [get_property IP_NAME $hw_proc_handle]
    set procname [get_property NAME $hw_proc_handle]
    set enable_sw_profile [get_property
CONFIG.enable_sw_intrusive_profiling $os_handle]
    set mb_exceptions false
    switch $proctype {
        "microblaze" {
        foreach entry [glob -nocomplain [file join $mbsrcdir *]] {
# Copy over only files that are not related to exception
handling.
# All such files have exception in their names.
file copy -force $entry "./src/"
}
    set need_config_file "true"
    set mb_exceptions [mb_has_exceptions $hw_proc_handle]
}
    "ps7_cortexa9" {
    set procdrv [get_sw_processor]
    set compiler [get_property CONFIG.compiler $procdrv]
    if {[string compare -nocase $compiler "armcc"] == 0} {
    set ccdir "./src/cortexa9/armcc"
    } else {
    set ccdir "./src/cortexa9/gcc"
    }
    foreach entry [glob -nocomplain [file join
    $cortexa9srcdir *]] {
    file copy -force $entry "./src/"
    }
    foreach entry [glob -nocomplain [file join $ccdir *]] {
    file copy -force $entry "./src/"
    }
    file delete -force "./src/armcc"
    file delete -force "./src/gcc"
    if {[string compare -nocase $compiler "armcc"] == 0} {
    file delete -force "./src/profile"
    set enable_sw_profile "false"
    }
```

```
    set file_handle [xopen_include_file "xparameters.h"]
    puts $file_handle "#include \"xparameters_ps.h\""
    puts $file_handle ""
    close $file_handle
    }
    "default" {puts "unknown processor type $proctype\n"}
}
```

MLD Parameter Descriptions

MLD Parameter Description Section

This section gives a detailed description of the constructs used in the MLD file.

Conventions

[ ] Denotes optional values.

< > Value substituted by the MLD writer.

Comments

Comments can be specified anywhere in the file. A "#" character denotes the beginning of a comment and all characters after the "#" right up to the end of the line are ignored. All white spaces are also ignored and semi-colons with carriage returns act as sentence delimiters.

OS or Library Definition

The OS or library section includes the OS or library name, options, dependencies, and other global parameters, using the following syntax:

```
option psf_version = <psf version number> BEGIN LIBRARY/OS <library/os
name> [option drc = <global drc name>] [option depends = <list of
directories>] [option help = <help file>] [option requires_interface =
<list of interface names>] PARAM <parameter description> [BEGIN CATEGORY
<name of category> <category description> END CATEGORY] BEGIN INTERFACE
<interface name> ..... END INTERFACE] END LIBRARY/OS
```

MLD Keywords

The keywords that are used in an MLD file are as follows:

BEGIN

The `BEGIN` keyword begins one of the following: `os`, `library`, `driver`, `block`, `category`, `interface`, and `array`.

END

The `END` keyword signifies the end of a definition block.

PSF_VERSION

Specifies the PSF version of the library.

DRC

Specifies the DRC function name. This is the global DRC function, which is called by the Vitis IDE configuration tool or the command-line tool. This DRC function is called once you enter all the parameters and MLD or MDD writers can verify that a valid OS, library, or driver can be generated with the given parameters.

Option

Specifies that the name following the keyword `option` is an option to the Vitis IDE tools.

OS

Specifies the type of OS. If it is not specified, then OS is assumed as standalone type of OS.

COPYFILES

Specifies the files to be copied for the OS, library, or driver. If `ALL` is used, then the tool copies all the OS, library, or driver files.

DEPENDS

Specifies the list of directories that needs to be compiled before the OS or library is built.

SUPPORTED_PERIPHERALS

Specifies the list of peripherals supported by the OS. The values of this option can be specified as a list, or as a regular expression. For example:

```
option supported_peripherals = (microblaze)
```

Indicates that the OS supports all versions of MicroBlaze. Regular expressions can be used in specifying the peripherals and versions. The regular expression (RE) is constructed as follows:

- Single-Character REs:
  - Any character that is not a special character (to be defined) matches itself.
  - A backslash (followed by any special character) matches the literal character itself. That is, this “escapes” the special character.

    - The special characters are: + * ? . [ ] ^ $
    - The period (.) matches any character except the new line. For example, .umpty matches both Humpty and Dumpty.
    - A set of characters enclosed in brackets ([]) is a one-character RE that matches any of the characters in that set. For example, [akm] matches either an "a", "k", or "m".
    - A range of characters can be indicated with a dash. For example, [a-z] matches any lowercase letter. However, if the first character of the set is the caret (^), then the RE matches any character except those in the set. It does not match the empty string. Example: [^akm] matches any character except "a", "k", or "m". The caret loses its special meaning if it is not the first character of the set.
- Multi-Character REs:
    - A single-character RE followed by an asterisk (*) matches zero or more occurrences of the RE. Thus, [a-z]* matches zero or more lower-case characters.
    - A single-character RE followed by a plus (+) matches one or more occurrences of the RE. Thus, [a-z]+ matches one or more lower-case characters.
    - A question mark (?) is an optional element. The preceeding RE can occur zero or once in the string, no more. Thus, xy?z matches either xyz or xz.
    - The concatenation of REs is a RE that matches the corresponding concatenation of strings. For example, [A-Z][a-z]* matches any capitalized word.
    - For example, the following matches a version of the axidma:

```
option supported_peripherals = (axi_dma_v[3-9]_[0-9][0-9]_[a-z]
axi_dma_v[3-9]_[0-9]);
```

LIBRARY_STATE

Specifies the state of the library. Following is the list of values that can be assigned to LIBRARY_STATE:

- **ACTIVE:** An active library. By default the value of LIBRARY_STATE is ACTIVE.
- **DEPRECATED:** This library is deprecated
- **OBSOLETE:** This library is obsolete and will not be recognized by any tools. Tools error out on an obsolete library and a new library should be used instead.

APP_COMPILER_FLAGS

This option specifies what compiler flags must be added to the application when using this library. For example:

```
option APP_COMPILER_FLAGS = “-D MYLIBRARY”
```

The Vitis IDE tools can use this option value to automatically set compiler flags automatically for an application.

APP_LINKER_FLAGS

This option specifies that linker flags must be added to the application when using a particular library or OS. For example:

```
option APP_LINKER_FLAGS = "-lxilkernel"
```

The Vitis IDE tools can use this value to set linker flags automatically for an application.

BSP

Specifies a boolean keyword option that can be provided in the MLD file to identify when an OS component is to be treated as a third party BSP. For example:

```
option BSP = true;
```

This indicates that the Vitis tools will offer this OS component as a board support package. If set to false, the component is handled as a native embedded software platform.

OS_STATE

Specifies the state of the operating system (OS). Following is the list of values that can be assigned to OS_STATE:

- **ACTIVE:** This is an active OS. By default the value of OS_STATE is ACTIVE.
- **DEPRECATED:** This OS is deprecated.
- **OBSOLETE:** This OS is obsolete and will not be recognized by the tools. Tools error out on an obsolete OS and a new OS must be specified.
- **OS_TYPE:** Specifies the type of OS. This value is matched with SUPPORTED_OS_TYPES of the driver MDD file for assigning the driver. Default is standalone.
- **REQUIRES_INTERFACE:** Specifies the interfaces that must be provided by other OSs, libraries, or drivers in the system.
- **REQUIRES_OS:** Specifies the list of OSs with which the specified library will work. For example:

  ```
  option REQUIRES_OS = (standalone xilkernel_v4_[0-9][0-9])
  ```

  The Vitis IDE tools use this option value to determine which libraries are offered for a given operating system choice. The values in the list can be regular expressions as shown in the example.

Note: This option must be used on libraries only.

- **HELP:** Specifies the `HELP` file that describes the OS, library, or driver.
- **DEP:** Specifies the condition that must be satisfied before processing an entity. For example to include a parameter that is dependent on another parameter (defined as a DEP, for dependent, condition), the DEP condition should be satisfied. Conditions of the form (operand1 OP operand2) are the only supported conditions.
- **INTERFACE:** Specifies the interfaces implemented by this OS, library, or driver. It describes the interface functions and header files used by the library/driver.

```
BEGIN INTERFACE <interface name>
option DEP=;<list of dependencies>;
PROPERTY HEADER=<name of header file where the function is
declared>;
FUNCTION NAME=<name of the interface function>, VALUE=<function
name of library/driver implementation> ;
END INTERFACE
```

- **HEADER:** Specifies the `HEADER` file in which the interface functions would be defined.
- **FUNCTION:** Specifies the `FUNCTION` implemented by the interface. This is a name-value pair in which name is the interface function name and value is the name of the function implemented by the OS, library, or driver.
- **CATEGORY:** Defines an unconditional block. This block gets included based on the default value of the category or if included in the MSS file.

```
BEGIN CATEGORY <category name>
PARAM name = <category name>, DESC=<param description>,
TYPE=<category type>,
DEFAULT=<default>, GUI_PERMIT=<value>, DEP = <condition>
option DEPENDS=<list of dependencies>, DRC=<drc name>, HELP=<help
file>;
<parameters or categories description>
END CATEGORY
```

  Nested categories are not supported through the syntax that specifies them. A category is selected in a MSS file by specifying the category name as a parameter with a boolean value `TRUE`. A category must have a PARAM with category name.

- **PARAM:** The MLD file has a simple <name = value> format for most statements. The PARAM keyword is required before every such `NAME`, `VALUE` pair. The format for assigning a value to a parameter is `param name = <name>, default = value`. The `PARAM` keyword specifies that the parameter can be overwritten in the MSS file.
- **PROPERTY:** Specifies the various properties of the entity defined with a `BEGIN` statement.
- **NAME:** Specifies the name of the entity in which it was defined. (Examples: `param` and `property`.) It also specifies the name of the library if it is specified with option.
- **VERSION:** Specifies the version of the library.

- **DESC:** Describes the entity in which it was defined. (Examples: `param` and `property`.)
- **TYPE:** Specifies the type for the entity in which it was defined. (Example: param) The following types are supported:
  - **bool:** Boolean (true or false)
  - **int:** integer
  - **string:** String value within " " (quotes)
  - **enum:** List of possible values that a parameter can take
  - **library:** Specify other library that is needed for building the library/driver
  - **peripheral_instance:** Specify other hardware drivers that is needed for building the library
- **DEFAULT:** Specifies the default value for the entity in which it was defined.
- **GUI_PERMIT:** Specifies the permissions for modification of values. The following permissions exist:
  - **NONE:** The value cannot be modified at all.
  - **ADVANCED_USER:** The value can be modified by all. The Vitis IDE does not display this value by default. This is displayed only for the advanced option in the Vitis IDE.
  - **ALL_USERS:** The value can be modified by all. The Vitis IDE displays this value by default. This is the default value for all the values. If GUI_PERMIT = NONE, the category is always active.
- **ARRAY:** ARRAY can have any number of PARAMs, and only PARAMs. It cannot have`CATEGORY` as one of the fields of an array element. The size of the array can be defined as one of the properties of the array. An array with default values specified in the default property leads to its size property being initialized to the number of values. If there is no size property defined, a size property is created before initializing it with the default number of elements. Each parameter in the array can have a default value. In cases in which size is defined with an integer value, an array of size elements would be created wherein the value of each element would be the default value of each of the parameters.

```
BEGIN ARRAY <array name>
PROPERTY desc = <array description> ;
PROPERTY size = <size of the array>;
PROPERTY default = <List of Values for each element based on the
size of the array>
# array field description as parameters
PARAM name = <name of parameter>, desc = "description of param",
type = <type of param>, default = <default value>
.....
END ARRAY
```

MLD Design Rule Check Section

```
proc mydrc { handle } { }
```

The DRC function could be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (which the tool builds using the hardware (XSA) and software (MSS) database files) to read the parameter values that you set. The handle is associated with the current library in the database. The DRC procedure can get the OS and library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameters.

For errors, DRC procedures call the Tcl error command error "error msg" that displays in an error page.

For warnings, DRC procedures return a string value that can be printed on the console.

On success, DRC procedures return without any value.

MLD Tool Generation (Generate) Section

```
proc mygenerate { handle } {
}
```

`Generate` could be any Tcl code that reads your parameters and generates configuration files for the OS or library. The configuration files can be C files, Header files, Makefiles, etc. The generate procedures can access (read-only) the Platform Specification Format database (which the tool builds using the MSS files) to read the parameter values of the OS or library that you set. The `handle` is a handle to the current OS or library in the database. The generate procedure can get the OS or library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameter.

Microprocessor Driver Definition (MDD)

Microprocessor Driver Definition Overview

A Microprocessor Driver Definition (MDD) file contains directives for customizing software drivers. This document describes the MDD format and the parameters that can be used to customize drivers.

Requirements

Each device driver has an MDD file and a Tool Command Language (Tcl) file associated with it. The MDD file is used by the Tcl file to customize the driver, depending on different options configured in the MSS file. For more information on the MSS file format, see Microprocessor Software Specification (MSS).

The driver source files and the MDD file for each driver must be located at specific directories in order to find the files and the drivers. This document describes the MDD format and the parameters that can be used to customize drivers.

MDD Driver Definition Files

Driver Definition involves defining a Data Definition file (MDD) and a Data Generation file (Tcl file).

- **Data Definition File:** The MDD file (`<driver_name>.mdd`) contains the configurable parameters. A detailed description of the parameters and the MDD format is described in MDD Parameter Description.
- **Data Generation File:** The second file (`<driver_name>.tcl`), with the filename being the same as the MDD filename) uses the parameters configured in the MSS file for the driver to generate data. Data generated includes but is not limited to generation of header files, C files, running DRCs for the driver, and generating executables. The Tcl file includes procedures that are called by the tool at various stages of its execution.

Various procedures in a Tcl file includes: the DRC (name of the DRC given in the MDD file), `generate` (tool defined procedure) called after driver files are copied, `post_generate` (tool defined procedure) called after generate is called on all drivers and libraries, and `execs_generate` called after the libraries and drivers are generated.

***Note*:** A driver does not require the data generation file (Tcl file).

MDD Format Specification

The MDD format specification involves the MDD file Format specification and the Tcl file Format specification which are described in the following subsections.

MDD File Format Specification

The MDD file format specification describes the parameters defined in the Parameter Description section. This data section describes configurable parameters in a driver. The format used to describe these parameters is discussed in MDD Parameter Description.

Tcl File Format Specification

Each driver has a Tcl file associated with the MDD file. This Tcl file has the following sections:

- **DRC Section:** This section contains Tcl routines that validate your driver parameters for consistency.
- **Generation Section:** This section contains Tcl routines that generate the configuration header and C files based on the driver parameters.

MDD Format Examples

This section explains the MDD format through an example of an MDD file and its corresponding Tcl file.

Example: MDD File

The following is an example of an MDD file for the uartlite driver.

```
option psf_version = 2.1;
```

`option` is a keyword identified by the tool. The option name following the `option` keyword is a directive to the tool to do a specific action. Here the `psf_version` of the MDD file is defined as 2.1. This is the only option that can occur before a `BEGIN DRIVER` construct.

```
BEGIN DRIVER uartlite
```

The `BEGIN DRIVER` construct defines the start of a driver named uartlite.

```
option supported_peripherals = (mdm axi_uartlite);
option driver_state = ACTIVE;
option copyfiles = all;
option VERSION = 3.0;
option NAME = uartlite;
```

The `NAME` option indicates the name of the driver. The `VERSION` option indicates the version of the driver. The `COPYFILES` option indicates the files to be copied for a “level” 0 uartlite driver.

```
BEGIN INTERFACE stdin
```

`BEGIN INTERFACE` defines an interface the driver supports. The interface name is `stdin`.

```
PROPERTY header = xuartlite_l.h;
FUNCTION name = inbyte, value = XUartLite_RecvByte;
END INTERFACE
```

An Interface contains a list of standard functions. A driver defining an interface should have values for the list of standard functions. It must also specify a header file in which all the function prototypes are defined.

PROPERTY defines the properties associated with the construct defined in the BEGIN construct. The header is a property with the value xuartlite_l.h, defined by the stdin interface. FUNCTION defines a function supported by the interface. The inbyte function of the stdin interface has the value XUartLite_RecvByte. This function is defined in the header file xuartlite_l.h.

```
BEGIN INTERFACE stdout
PROPERTY header = xuartlite_l.h;
FUNCTION name = outbyte, value = XUartLite_SendByte;
END INTERFACE
BEGIN INTERFACE stdio
PROPERTY header = xuartlite_l.h;
FUNCTION name = inbyte, value = XUartLite_RecvByte;
FUNCTION name = outbyte, value = XUartLite_SendByte;
END INTERFACE
```

END is used with the construct name that was used in the BEGIN statement. Here END is used with BLOCK and DRIVER constructs to indicate the end of each BLOCK and DRIVER construct.

Example: Tcl File

The following is the uartlite.tcl file corresponding to the uartlite.mdd file described in the previous section. The “uartlite_drc” procedure would be invoked for the uartlite driver while running DRCs for drivers. The generate routine generates constants in a header file and a c file for uartlite driver, based on the driver definition segment in the MSS file.

```
proc generate {drv_handle} {
::hsi::utils::define_include_file $drv_handle "xparameters.h"
"XUartLite" "NUM_INSTANCES" "C_BASEADDR"
"C_HIGHADDR" "DEVICE_ID" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY"
"C_DATA_BITS"
::hsi::utils::define_config_file $drv_handle "xuartlite_g.c"
"XUartLite" "DEVICE_ID" "C_BASEADDR"
"C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY" "C_DATA_BITS"
::hsi::utils::define_canonical_xpars $drv_handle "xparameters.h"
"UartLite" "DEVICE_ID" "C_BASEADDR"
"C_HIGHADDR" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY" "C_DATA_BITS"
}
```

MDD Parameter Description

This section gives a detailed description of the constructs used in the MDD file.

Conventions

[ ]: Denotes optional values.

< >: Value substituted by the MDD writer.

Comments

Comments can be specified anywhere in the file. A pound (#) character denotes the beginning of a comment, and all characters after it, right up to the end of the line, are ignored. All white spaces are also ignored and semicolons with carriage returns act as sentence delimiters.

Driver Definition

The driver section includes the driver name, options, dependencies, and other global parameters, using the following syntax:

```
option psf_version = <
psf version number>
BEGIN DRIVER <driver name>
[option drc = <global drc name>]
[option depends = <list of directories>]
[option help = <help file>]
[option requires_interface = <list of interface names>
]
PARAM <parameter description>
[BEGIN BLOCK,dep = <condition>
.......
END BLOCK]
[BEGIN INTERFACE <interface name>
.......
END INTERFACE]
END DRIVER
```

MDD Keywords

The keywords that are used in an MDD file are as follows:

Begin

The BEGIN keyword begins with one of the following: library, drive, block, category, or interface.

END

The END keyword signifies the end of a definition block.

PSF_VERSION

Specifies the PSF version of the library.

DRC

Specifies the DRC function name. This is the global DRC function that is called by the Vitis IDE configuration tool or the command-line tool. This DRC function is called when you enter all the parameters and the MLD or MDD writers can verify that a valid library or driver can be generated with the given parameters.

option

Specifies the name following the keyword option is an option to the tool. The following five options are supported: COPYFILES, DEPENDS, SUPPORTED_PERIPHERALS, and DRIVER_STATE.

SUPPORTED_OS_TYPES

Specifies the list of supported OS types. If it is not specified, then driver is assumed as standalone driver.

COPYFILES

Specifies the list of files to be copied for the driver. If ALL is specified as the value, the tool copies all the driver files.

DEPENDS

Specifies the list of directories on which a driver depends for compilation.

SUPPORTED_PERIPHERALS

Specifies the list of peripherals supported by the driver. The values of this option can be specified as a list or as a regular expression. The following example indicates that the driver supports all versions of opb_jtag_uart and the opb_uartlte_v1_00_b version:

```
option supported_peripherals = (xps_uartlite_v1_0, xps_uart16550)
```

Regular expressions can be used in specifying the peripherals and versions. The regular expression (RE) is constructed as described below.

Single-Character REs

- Any character that is not a special character (to be defined) matches itself.
- A backslash (followed by any special character) matches the literal character itself. That is, it escapes the special character.
- The special characters are: + * ? . [ ] ^ $
- The period matches any character except the newline. For example, .umpty matches both Humpty and Dumpty.
- A set of characters enclosed in brackets ([]) is a one-character RE that matches any of the characters in that set. For example, [akm]matches an a, k, or m. A range of characters can be indicated with a dash. For example, [a-z] matches any lower-case letter.

However, if the first character of the set is the caret (^), then the RE matches any character except those in the set. It does not match the empty string. For example, [^akm] matches any character except a, k, or m. The caret loses its special meaning if it is not the first character of the set.

Multi-Character REs

- A single-character RE followed by an asterisk (*) matches zero or more occurrences of the RE. Therefore, [a-z]* matches zero or more lower-case characters.
- A single-character RE followed by a plus (+) matches one or more occurrences of the RE. Therefore, [a-z]+ matches one or more lower-case characters.
- A question mark (?) is an optional element. The preceding RE can occur no times or one time in the string. For example, xy?z matches either xyz or xz.
- The concatenation of REs is an RE that matches the corresponding concatenation of strings. For example, [A-Z][a-z]* matches any capitalized word.

The following example matches any version of xps_uartlite, xps_uart16550, and mdm.

```
option supported_peripherals = (xps_uartlite_v[0-9]+_[1-9][0-9]_[a-z]
xps_uart16550 mdm);
```

DRIVER_STATE

Specifies the state of the driver. The following are the list of values that can be assigned to DRIVER_STATE:

- **ACTIVE:** This is an active driver. By default the value of DRIVER_STATE is ACTIVE.
- **DEPRECATED:** This driver is deprecated and is scheduled to be removed.
- **OBSOLETE:** This driver is obsolete and is not recognized by any tools. Tools error out on an obsolete driver, and a new driver should be used instead.

REQUIRES_INTERFACE

Specifies the interfaces that must be provided by other libraries or drivers in the system.

HELP

Specifies the help file that describes the library or driver.

DEP

Specifies the condition that needs to be satisfied before processing an entity. For example, to enter into a BLOCK, the DEP condition should be satisfied. Conditions of the form (operand1 OP operand2) are supported.

BLOCK

Specifies the block is to be entered into when the DEP condition is satisfied. Nested blocks are not supported.

INTERFACE

Specifies the interfaces implemented by this library or driver and describes the interface functions and header files used by the library or driver.

```
BEGIN INTERFACE <interface name>
option DEP=<list of dependencies>;
PROPERTY HEADER=<name of header file where the function is declared>
;
FUNCTION NAME=<name of interface function>, VALUE=<function name
of library/driver implementation> ;
END INTERFACE
```

HEADER

Specifies the header file in which the interface functions would be defined.

FUNCTION

Specifies the function implemented by the interface. This is a name-value pair where name is the interface function name and value is the name of the function implemented by the library or driver.

PARAM

Generally, the MLD/MDD file has a name = value format for statements. The PARAM keyword is required before every such NAME, VALUE pair. The format for assigning a value to a parameter is param name = <name>, default= value. The PARAM keyword specifies that the parameter can be overwritten in the MSS file.

DTGPARAM

The DTGPARAM keyword is specially used for the device-tree specific parameters that can be configured. Driver defines these DTGPARAMs if it needs to dump any parameters in the Tool DTG generated DTS file.

PROPERTY

Specifies the various properties of the entity defined with a BEGIN statement.

NAME

Specifies the name of the entity in which it was defined (example: PARAM, PROPERTY). It also specifies the name of the driver if it is specified with option.

VERSION

Specifies the version of the driver.

DESC

Describes the entity in which it was defined (example: PARAM, PROPERTY).

TYPE

Specifies the type for the entity in which it was defined (example: PARAM). The following are the supported types:

- bool: Boolean (true or false)

  int: Integer

  string: String value within “ " (quotes).

  enum: List of possible values, that this parameter can take.

  library: Specify other library that is needed for building the library or driver.

  peripheral_instance: Specify other hardware drivers needed for building the library or driver. Regular expressions can be used to specify the peripheral instance. Refer to SUPPORTED_PERIPHERALS in MLD Keywords for more details about regular expressions.

DEFAULT

Specifies the default value for the entity in which it was defined.

GUI_PERMIT

Specifies the permissions for modification of values. The following permissions exist:

- **NONE:** The value can not be modified at all.
- **ADVANCED_USER:** The value can be modified by all. The Vitis IDE does not display this value by default. It is displayed only as an advanced option in the Vitis IDE.
- **ALL_USERS:** The value can be modified by all. The Vitis IDE displays this value by default. This is the default value for all the values. If GUI_PERMIT = NONE, the category is always active.

MDD Design Rule Check (DRC) Section

```
proc mydrc { handle }
```

The DRC function can be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (built by the tool using the hardware (XSA) and software (MSS) database files) to read the parameter values you set. The "handle" is a handle to the current driver in the database. The DRC procedure can get the driver parameters from this handle. It can also get any other parameter from the database by first requesting a handle and then using the handle to get the parameters.

- For errors, DRC procedures call the Tcl error command error "error msg" that displays in an error page.
- For warnings, DRC procedures return a string value that can be printed on the console.
- On success, DRC procedures return without any value.

MDD Driver Generation (Generate) Section

```
proc mygenerate { handle }
```

`generate` could be any Tcl code that reads your parameters and generates configuration files for the driver. The configuration files can be C files, Header files, or Makefiles. The `generate` procedures can access (read-only) the Platform Specification Format database (built by the tool using the MSS files) to read the parameter values of the driver that you set. The `handle` is a handle to the current driver in the database. The generate procedure can get the driver parameters from this handle. It can also get any other parameters from the database by requesting a handle and then using the handle to get the parameter.

Custom Driver

This section demonstrates how to hand-off a custom driver associated with an IP(driver files are specified in IPXACT file of the IP component) and access the driver information in HSI as well as associate the driver with IP during BSP generation. For more information on packaging IP with custom driver, refer to *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118).

An example design of an IP with custom driver specified in its IPXACT definition.

Figure 5: **Example Design with an IP with Custom Driver**

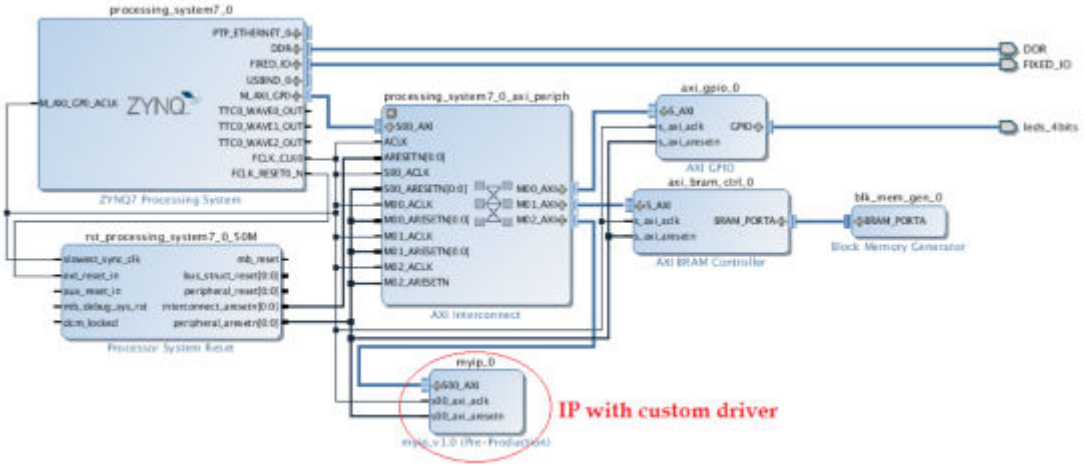
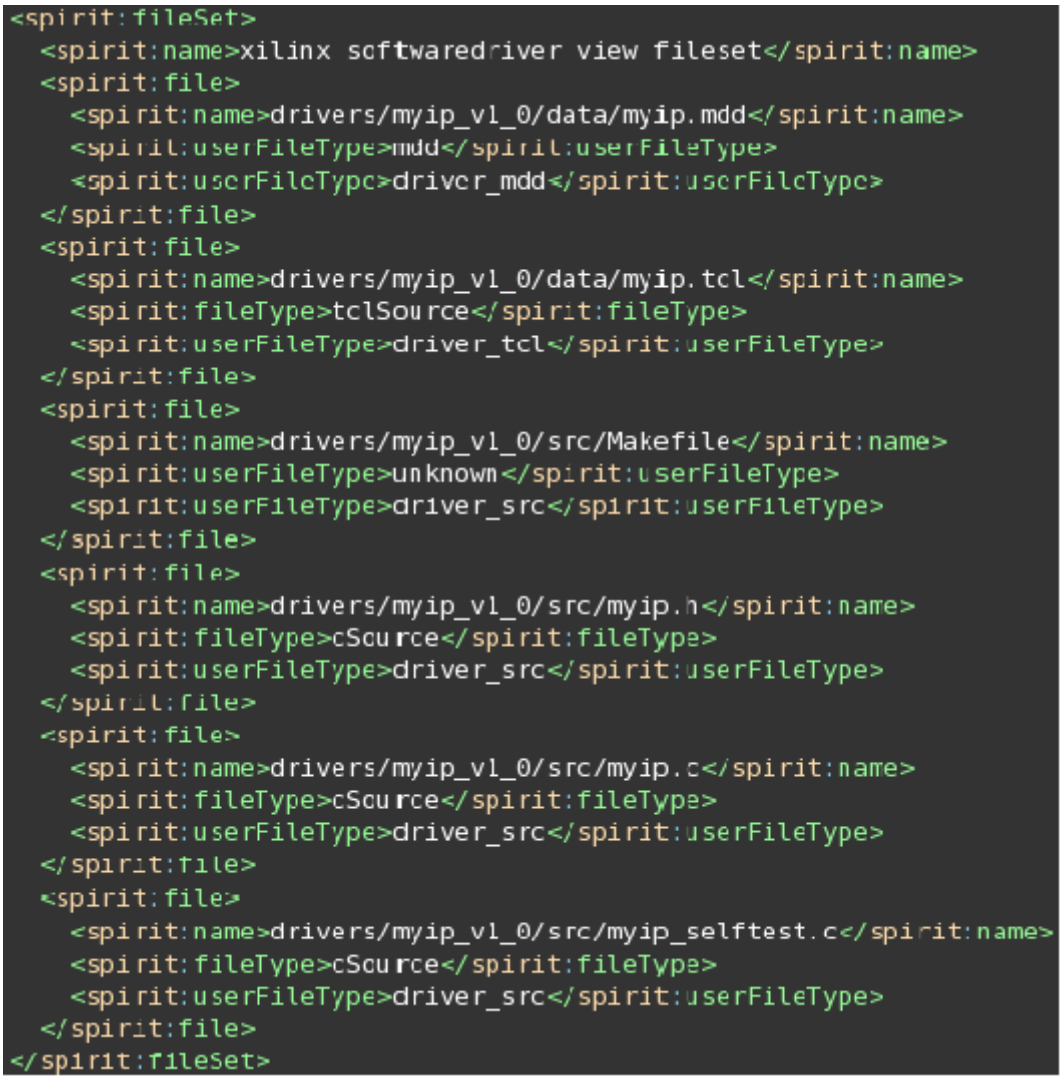


Figure 6: **Custom Driver Specified in IPXACT Specification of an IP**



Custom Driver Specified in IPXACT Specification of an IP

Run Vivado hardware hand-off flow either in Pre-Synth or Post-Bitstream mode. The custom driver for each IP is packaged in an XSA.

Open the hardware design with custom drivers.

```
hsi::open_hw_design ./base_zynq_design_wrapper.xsa
                    base_zynq_design_wrapper
```

Create a software design

```
hsi::create_sw_design swdesign -proc ps7_cortexa9_0 -os standalone
                    Swdesign
```

Check if the custom drivers are assigned to respective IP cores or not

```
join [hsi::get_drivers ] \n
                    axi_bram_ctrl_0
                    axi_gpio_0
                    myip_0
```

Check the custom driver properties

```
common::report_property [ hsi::get_drivers myip*]
```

Table 7: **Example Table**

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	driver
HW_INSTANCE	string	true	true	myip_0
NAME	string	false	true	myip
VERSION	string	false	true	1.0

Generate BSP. BSP source code including custom driver sources will be dumped to the bsp_out #directory

```
hsi::generate_bsp -dir bsp_out
                    base_zynq_design_wrapper
                    ls ./bsp_out/ps7_cortexa9_0/libsrc/
                    . . .
                    myip_v1_0
                    . . .
```

Microprocessor Application Definition (MAD)

Microprocessor Application Definition Overview

A MAD file contains directives for customizing software application. This section describes the Microprocessor Application Definition (MAD) format, Platform Specification Format 2.1.0. and the parameters that can be used to customize applications.

Requirements

Each application has an MAD file and a Tool Command Language (Tcl) file associated with it.

The MAD file is used by Hsi to recognize it as an application and to consider its configuration while generating the application sources. The MAD file for each application must be located in its data directory.

Microprocessor Application Definition Files

Application Definition involves defining a Microprocessor Application Definition file (MAD) and a Data Generation file (Tcl file).

Application Definition File

The MAD file (`<application_name>.mad`) contains the name, description and other configurable parameters. A detailed description of the various parameters and the MAD format is described in MAD Format Specification.

Data Generation File

The second file (`<application_name>.tcl`, with the filename being the same as the MAD filename) uses the parameters in the MAD file for the application to generate data.

Data generated includes, but is not limited to, generation of header files, C files, running DRCs for the application and generating executables. The Tcl file includes procedures that are called by the tool at various stages of its execution. Various procedures in a Tcl file includes the following:

- `DRC` (swapp_is_supported_hw, swapp_is_supported_sw)
- `swapp_generate` (tool defined procedure) called after application source files are copied

MAD Format Specification

The MAD format specification involves the MAD file format specification and the Tcl file format specification.

MAD File Format Specification

The MAD file format specification describes the parameters using a sample MAD file and its corresponding Tcl file.

The following example shows a MAD file for a sample application called my_application.

```
option psf_version = 2.1;
```

`option` is a keyword identified by the tool. The option name following the `option` keyword is a directive to the tool to do a specific action.

The `psf_version` of the MAD file is defined to be 2.1 in this example. This is the only option that can occur before a BEGIN APPLICATION construct.

```
BEGIN APPLICATION my_application
```

The `BEGIN APPLICATION` construct defines the start of an application named `my_application`.

```
option NAME = myapplication
                    option DESCRIPTION = "My custom application"
                    END APPLICATION
```

Note: The application NAME should match the return value of the Tcl process swapp_get_name in the application Tcl file described above.

Tcl File Format Specification

Each application has a Tcl file associated with the MAD file. This Tcl file has the following sections:

- *DRC Section*: This section contains Tcl routines that validate your hardware and software instances and their configuration needed for the application.
- *Generation Section*: This section contains Tcl routines that generate the application header and C files based on the hardware and software configuration.

MAD Format Example

This section explains the MAD format through an example MAD file and its corresponding Tcl file.

Example: MAD File

The following is an example of an MAD file for a sample application called my_application.

```
option psf_version = 2.1;
```

`option` is a keyword identified by the tool. The option name following the `option` keyword is a directive to the tool to do a specific action.

The `psf_version` of the MAD file is defined to be 2.1 in this example. This is the only option that can occur before a BEGIN APPLICATION construct .

```
BEGIN APPLICATION my_application
```

The `BEGIN APPLICATION` construct defines the start of an application named `my_application`.

```
option NAME = myapplication
                    option DESCRIPTION = "My custom application"
                    END APPLICATION
```

Note: Application NAME should match the return value of Tcl proc swapp_get_name in application Tcl file described above.

HSI Commands

This section contains all hardware and software interface Tcl commands, arranged alphabetically.

common::get_property

Description

Get properties of object.

Syntax

```
get_property [-min] [-max] [-quiet] [-verbose] <name> <object>
```

Returns

Property value.

Usage

Name	Description
[-min]	Return only the minimum value
[-max]	Return only the maximum value
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of property whose value is to be retrieved
<object>	Object to query for properties

Categories

Object, PropertyAndParameter

Description

Gets the current value of the named property from the specified object or objects. If multiple objects are specified, a list of values is returned.

If the property is not currently assigned to the object, or is assigned without a value, then the `get_property` command returns nothing, or the null string. If multiple objects are queried, the null string is added to the list of values returned.

This command returns a value, or list of values, or returns an error if it fails.

Arguments

`-min` - (optional) When multiple objects are specified, this option examines the values of the named property, and returns the smallest value from the list of objects. Numeric properties are sorted by value. All other properties are sorted as strings.

`-max` - (optional) When multiple objects are specified, this option examines the values of the named property, and returns the largest value from the list of objects. Numeric properties are sorted by value. All other properties are sorted as strings.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

`name` - (required) The name of the property to be returned. The name is not case sensitive.

`object` - (required) One or more objects to examine for the specified property.

Examples

Get the NAME property from the specified cell:

```
common::get_property NAME [lindex [get_cells] 0]
```

Get the BOARD property from the current hardware design:

```
common::get_property BOARD [current_hw_design]
```

common::report_property

Description

Report properties of object.

Syntax

```
report_property [-all] [-class <arg>] [-return_string] [-file <arg>] [-
append] [-regexp] [-quiet] [-verbose] [<object>] [<pattern>]
```

Returns

Property report.

Usage

Name	Description
[-all]	Report all properties of object even if not set
[-class]	Object type to query for properties. Not valid with <object>
[-return_string]	Set the result of running report_property in the Tcl interpreter's result variable
[-file]	Filename to output result to. Send output to console if -file is not used
[-append]	Append the results to file; do not overwrite the results file
[-regexp]	Pattern is treated as a regular expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<object>]	Object to query for properties
[<pattern>]	Pattern to match properties against Default: *

Categories

Object, PropertyAndParameter, Report

Description

Gets the property name, property type, and property value for all of the properties on a specified object, or class of objects.

***Note*:** list_property also returns a list of all properties on an object, but does not include the property type or value.

You can specify objects for report_property using the get_* series of commands to get a specific object. You can use the lindex command to return a specific object from a list of objects:

```
    report_property [lindex [get_cells] 0]
```

However, if you are looking for the properties on a class of objects, you should use the -class option instead of an actual object.

This command returns a report of properties on the object, or returns an error if it fails.

Arguments

-all> - (optional) Return all of the properties for an object, even if the property value is not currently defined.

`-class` <arg>- (optional) Return the properties of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

***Note*:** -class cannot be used together with an <object>

`-return_string`- (optional) Directs the output to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

`-file`<arg>- (optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless -append is also specified.

***Note*:** If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

`-append` - (optional) Append the output of the command to the specified file rather than overwriting it.

***Note*:** The -append option can only be used with the `-file` option.

`-regexp`- (optional) Specifies that the search <pattern> is written as a regular expression.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

<object> - (optional) A single object on which to report properties.

***Note*:** If you specify multiple objects you will get an error.

<pattern> - (optional) Match the available properties on the <object> or -class against the specified search pattern. The <pattern> applies to the property name, and only properties matching the specified pattern will be reported. The default pattern is the wildcard `*` which returns a list of all properties on the specified object.

***Note*:** The search pattern is case sensitive, and most properties are UPPER case.

Examples

The following example returns all properties of the specified object:

```
common::report_property -all [get_cells microblaze_0]
```

To determine which properties are available for the different design objects supported by the tool, you can use multiple `report_property` commands in sequence. The following example returns all properties of the specified current objects:

```
common::report_property -all [current_hw_design]
```

```
common::report_property -all [current_sw_design]
```

hsi::close_hw_design

Description

Close a hardware design.

Syntax

```
close_hw_design [-quiet] [-verbose] <name>
```

Returns

Returns nothing, error message if failed.

Usage

Name	Description
[`-quiet`]	Ignore command errors
[`-verbose`]	Suspend message limits during command execution
<name>	Name of design to close

Categories

Hardware

Description

Closes the hardware design in the HSM active session. Design modification is not allowed in the current release, otherwise it will prompt to save the design prior to closing.

Arguments

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

<name> - The name of the hardware design object to close.

Examples

Close the current hardware design object:

```
hsi::close_hw_design [current_hw_design]
```

Close the specified hardware design object:

```
hsi::close_hw_design design_1_imp
```

hsi::create_dt_node

Description

Create a DT node.

Syntax

```
create_dt_node -name <arg> [-unit_addr <arg>] [-label <arg>] [-objects
<args>] [-quiet] [-verbose]
```

Returns

DT node object. Returns nothing if the command fails.

Usage

Name	Description
`-name`	Child DT node name
[`-unit_addr`]	Unit address of node
[`-label`]	Label of node
[`-objects`]	List of nodes
[`-quiet`]	Ignore command errors
[`-verbose`]	Suspend message limits during command execution

Categories

DeviceTree

Description

Create a new DT node and add to the current DT tree.

If successful, this command returns the name of the DT node created where name is represented as "node_label"+"node_name"+"@unit_address". Otherwise it returns an error.

Arguments

`-name` - The name of the node to be created.

`-label` - The label of the node to represent in generated dtsi file.

`--unit_addr` - The unit address of the node to represent in generated dtsi file.

`-objects` - The list of node objects where the newly created node will be a child to all specified nodes.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

Examples

Create a new DT node amba with lable axi_interconnect and unit_addr 0x000 in the current DT tree:

```
hsi::create_dt_node -name amba -label axi_interconnect -unit_addr 0x0000
```

```
hsi::create_dt_node -name amba -label axi_interconnect -unit_addr 0x0000 -
objects [get_dt_nodes -of_objects\>
```

hsi::create_dt_tree

Description

Create a DT tree.

Syntax

```
create_dt_tree -dts_file <arg> [-dts_version <arg>] [-quiet] [-verbose]
```

Returns

Tree object. Returns nothing if the command fails.

Usage

Name	Description
`-dts_file`	dts file name
[`-dts_version`]	dts version
[`-quiet`]	Ignore command errors
[`-verbose`]	Suspend message limits during command execution

Categories

DeviceTree

Description

Create a new DT tree add to the current HSI session.

If successful, this command returns the name of the DT tree created. Otherwise it returns an error.

Arguments

`-dts_file` - The DT tree name or file name targeted for the output DTSI file.

`-dts_version` - The DTS version of the DTSI file.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

Examples

Create a new DT tree `pl.dtsi` and add the tree to the current session:

```
hsi::create_dt_tree -dts_file pl.dtsi -dts_version /dts-v1/
```

```
hsi::create_dt_tree -dts_file system.dts -dts_version /dts-v3/ -
header "include pl.dtsi, include ps.dtsi"
```

```
hsi::create_dt_tree -dts_file ps.dtsi -dts_version /dts-v3/ -
header "PS system info"
```

hsi::get_cells

Description

Get a list of cells.

Syntax

```
get_cells [-regexp] [-filter <arg>] [-hierarchical] [-of_objects <args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Cell objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-hierarchical]	Get cells from all levels of hierarchical cells
[-of_objects]	Get 'cell' objects of these types: 'hw_design port bus_intf net intf_net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of IP instance objects in the current design that match a specified search pattern. The default command returns a list of all IP instances in the design.

***Note*:** To improve memory and performance, the commands return a container list of a single type of objects (e.g. cells, nets, or ports). You can add new objects to the list (using lappend for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

Note: The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For cell objects, "IP_TYPE", and "IP_NAME" are some of the properties you can use to filter results. The following gets cells with an IP_TYPE of "PROCESSOR" and with names containing "ps7":

```
get_cells * -filter {IP_TYPE == PROCESSOR && NAME !~ "*ps7*"}
```

`-hierarchical` - (optional) Get cells from all levels of hierarchical cells .

`-of_objects` <arg> - (optional) Get the cells connected to the specified pins, timing paths, nets, bels, clock regions, sites or DRC violation objects.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns> - (optional) Match cells against the specified patterns. The default pattern is the wildcard `*` which gets a list of all cells in the project. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns list of processor instances:

```
hsi::get_cells -filter { IP_TYPE == "PROCESSOR" }
```

This example gets a list of properties and property values attached to the second object of the list returned by get_cells:

```
common::report_property [lindex [get_cells] 1]
```

Note: If there are no cells matching the pattern you get a warning.

hsi::get_dt_nodes

Description

Get a list of DT node objects.

Syntax

```
get_dt_nodes [-hier] [-regexp] [-filter <arg>] [-of_objects <args>] [-
quiet] [-verbose] [<patterns>...]
```

Returns

Node objects. Returns nothing if the command fails.

Usage

Name	Description
[-hier]	List of nodes in the current tree.
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get '' objects of these types: 'dtsNode dtsTree'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Name	Description
[<patterns>]	Match cell names against patterns Default: *

Categories

DeviceTree

Description

Gets a list of DT nodes created under a DT tree in the current HSI session that match a specified search pattern. The default command gets a list of all root DT nodes in the current DT tree.

Arguments

`-of_objects` <arg> - (optional) Gets all nodes of DTSNode and DTSTree

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_dt_nodes` or `get_dt_trees`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>.

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

Note: The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns> - (optional) Match nodes against the specified patterns. The default pattern is the wildcard `*` which gets a list of all root nodes in the current DT tree. More than one pattern can be specified to find multiple nodes based on different search criteria.

***Note*:** You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of root nodes attached to the specified DT tree:

```
hsi::get_dt_nodes -of_objects [lindex [get_dt_trees] 1]
```

***Note*:** If there are no nodes matching the pattern, the tool will return empty.

The following example gets a list of all nodes in the current DT tree:

```
hsi::get_dt_nodes -hier
```

***Note*:** If there are no nodes matching the pattern, the tool will return empty.

The following example gets a list of nodes created under a root node:

```
hsi::get_dt_nodes -of_objects [current_dt_tree]
```

***Note*:** If there are no nodes matching the pattern, the tool will return empty.

hsi::get_dt_trees

Description

Get a list of dts trees created.

Syntax

```
get_dt_trees [-regexp] [-filter <arg>] [-quiet] [-verbose] [<patterns>...]
```

Returns

DTS tree objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match tree names against patterns Default: *

Categories

DeviceTree

Description

Gets a list of DT trees created in the current HSI session that match a specified search pattern. The default command gets a list of all open DT trees in the HSI session.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

***Note*:** The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

***Note*:** The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For the "DT tree" object you can use the"DTS_FILE_NAME" property to filter results. The following gets dt trees that do NOT contain the "pl.dtsi" substring within their name:

```
get_dt_trees * -filter {NAME !~ "*pl.dtsi*"}
```

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

<patterns> - (optional) Match DT trees against the specified patterns. The default pattern is the wildcard `*` which gets all DT trees. More than one pattern can be specified to find multiple trees based on different search criteria.

Examples

Get all created DT trees in the current session:

```
hsi::get_dt_trees
```

hsi::get_intf_nets

Description

Get a list of interface nets.

Syntax

```
get_intf_nets [-regexp] [-filter <arg>] [-boundary_type <arg>] [-
hierarchical] [-of_objects <args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Interface Net objects. Returns nothing if the command fails.

Usage

Name	Description
[`-regexp`]	Patterns are full regular expressions
[`-filter`]	Filter list with expression
[`-boundary_type`]	Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper
[`-hierarchical`]	Get interface nets from all levels of hierarchical cells
[`-of_objects`]	Get 'intf_net' objects of these types: 'hw_design cell bus_intf'.
[`-quiet`]	Ignore command errors
[`-verbose`]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of interface nets in the current hardware design that match a specified search pattern. The default command gets a list of all interface nets in the subsystem design.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

***Note*:** The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

***Note*:** The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For hardware design nets you can use the "NAME" property to filter results.

`-hierarchical` - (optional) Get interface nets from all levels of hierarchical cells.

`-boundary_type` - (optional) Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations.

`-of_objects` <args> - (optional) Get a list of the nets connected to the specified IP integrator subsystem cell, pin, or port objects.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search pattern.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns> - (optional) Match hardware design interface nets against the specified patterns. The default pattern is the wildcard `*` which returns a list of all interface nets in the current IP integrator subsystem design. More than one pattern can be specified to find multiple nets based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the interface net attached to the specified pin of an hardware design, and returns the net:

```
hsi::get_intf_nets -of_objects [get_pins aclk]
```

Note: If there are no interface nets matching the pattern, you get a warning.

hsi::get_intf_pins

Description

Get a list of interface pins.

Syntax

```
get_intf_pins [-regexp] [-filter <arg>] [-hierarchical] [-of_objects
<args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Interface pin objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-hierarchical]	Get interface pins from all levels of hierarchical cells
[-of_objects]	Get 'bus_intf' objects of these types: 'hw_design cell port intf_net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of pin objects in the current design that match a specified search pattern. The default command gets a list of all pins in the design.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

***Note*:** The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

***Note*:** The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For the interface pins, "NAME" and "TYPE" are some of the properties you can use to filter results. The following gets slave interface pins that do NOT contain the "S_AXI" substring within their name:

```
get_intf_pins * -filter {TYPE == SLAVE && NAME !~ "*S_AXI*"}
```

`-hierarchical` - (optional) Get interface pins from all levels of hierarchical cells.

`-of_objects <arg>` - (optional) Get the pins connected to the specified cell, clock, timing path, or net; or pins associated with specified DRC violation objects.

***Note*:** The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>

`-match_style [sdc | ucf]` - (optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

`patterns` - (optional) Match pins against the specified patterns. The default pattern is the wildcard `*` which gets a list of all pins in the project. More than one pattern can be specified to find multiple pins based on different search criteria.

***Note*:** You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of pins attached to the specified cell:

```
hsi::get_intf_pins -of_objects [lindex [get_cells] 1]
```

***Note*:** If there are no pins matching the pattern, the tool will return a warning.

hsi::get_intf_ports

Description

Get a list of interface ports.

Syntax

```
get_intf_ports [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Interface Port objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'bus_intf' objects of these types: 'hw_design port intf_net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of interface port objects in the current hardware subsystem design that match a specified search pattern. The default command gets a list of all interface ports in the subsystem design.

The external connections in an IP subsystem design are ports, or interface ports. The external connections in an IP integrator cell, or hierarchical module, are pins and interface pins. Use the `get_pins` and `get_intf_pins` commands to select the pin objects.

***Note*:** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using lappend for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

Note: The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For IP subsystem interface ports, "DIRECTION", and "NAME" are some of the properties you can use to ilter results.

`-of_objects <arg>` - (optional) Get the interface ports connected to the specified IP subsystem interface nets returned by get_intf_nets.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

`patterns` - (optional) Match interface ports against the specified patterns. The default pattern is the wildcard `*` which gets a list of all interface ports in the subsystem design. More than one pattern can be specified to find multiple interface ports based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the interface ports in the subsystem design that operate in Master mode:

```
hsi::get_intf_ports -filter {MODE=="master"}
```

Note: If there are no interface ports matching the pattern, the tool will return a warning.

hsi::get_mem_ranges

Description

Get a list of memory ranges.

Syntax

```
get_mem_ranges [-regexp] [-filter <arg>] [-hierarchical] [-of_objects
<args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Memory range objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-hierarchical]	Get memory ranges from all levels of hierarchical cells
[-of_objects]	Get 'mem_range' objects of these types: 'cell'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Get a list of slaves of the processor in the current hardware design.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

Note: The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

`-hierarchical` - (optional) Get memory ranges from all levels of hierarchical cells.

`-of_objects <arg>` - (optional) Get the slaves of the specified object.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

`patterns` - (optional) Match address segments against the specified patterns. The default pattern is the wildcard `*` which gets a list of all address segments in the current IP subsystem design. More than one pattern can be specified to find multiple address segments based on different search criteria.

***Note*:** You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the slaves of the processor:

```
hsi::get_mem_ranges
```

```
hsi::get_mem_ranges -of_objects [lindex [get_cells -
filter {IP_TYPE==PROCESSOR} ] 0]
```

***Note*:** If there are no objects matching the pattern, you get a warning.

hsi::get_nets

Description

Get a list of nets.

Syntax

```
get_nets [-regexp] [-filter <arg>] [-boundary_type <arg>] [-hierarchical]
[-of_objects <args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Net objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-boundary_type]	Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper
[-hierarchical]	Get nets from all levels of hierarchical cells
[-of_objects]	Get 'net' objects of these types: 'hw_design cell port'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of nets in the current hardware design that match a specified search pattern. The default command gets a list of all nets in the subsystem design.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

***Note*:** The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

***Note*:** The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For the "hardware design" object you can use the "NAME" property to filter results.

`-boundary_type` - (optional) Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper.

`-hierarchical` - (optional) Get nets from all levels of hierarchical cells.

`-of_objects` - (optional) Get 'net' objects of these types: 'hw_design cell port'.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

`<patterns` - (optional) Match hardware design nets against the specified patterns. The default pattern is the wildcard `*` which returns a list of all nets in the current IP integrator subsystem design. More than one pattern can be specified to find multiple nets based on different search criteria.

***Note*:** You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the net attached to the specified pin of an hardware design module, and returns both the net:

```
hsi::get_nets -of_objects [get_pins aclk]
```

***Note*:** If there are no nets matching the pattern, you get a warning.

hsi::get_nodes

Description

Get a list of child nodes.

Syntax

```
get_nodes [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet] [-
verbose] [<patterns>...]
```

Returns

Node objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression

Name	Description
[-of_objects]	Get 'node' objects of these types: 'driver sw_proc os node'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Software

Description

Get a list of nodes in drivers/os/nodes in the current software design.

A node can have child nodes in it.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

***Note*:** The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

***Note*:** The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

The following gets nodes that matches NAME and PARENT within their name:

```
get_nodes -filter {NAME==clkc && PARENT == ps7_slcr_0}
```

`-of_objects <arg>` - (optional) Get 'node' objects of these types: 'sw_driver', 'sw_os', 'sw_proc', 'sw_node'.

***Note*:** The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_nodes`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the `set_msg_config` command.

`patterns` - (optional) Match sotfware design cells against the specified patterns. The default pattern is the wildcard `*` which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

***Note*:** You must enclose multiple search patterns in braces, {}, to present the list as a single element.

Examples

The following example gets a list of nodes that include the specified driver in the software design:

```
hsi::get_nodes -of_objects [get_drivers ps7_uart_0]
```

The following example gets a list of all nodes of OS:

```
hsi::get_nodes -of_objects [get_os]
```

hsi::get_pins

Description

Get a list of pins.

Syntax

```
get_pins [-regexp] [-filter <arg>] [-hierarchical] [-of_objects <args>] [-
quiet] [-verbose] [<patterns>...]
```

Returns

Pin objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-hierarchical]	Get pins from all levels of hierarchical cells
[-of_objects]	Get 'port' objects of these types: 'hw_design cell bus_intf net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of pin objects on the current hardware design that match a specified search pattern. The default command gets a list of all pins in the subsystem design.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

***Note*:** The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For pins, "DIR" and "TYPE" are some of the properties you can use to filter results. The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

`-hierarchical` - (optional) Get pins from all levels of hierarchical cells.

`-of_objects <arg>` - (optional) Get the pins connected to the specified IP subsystem cell or net.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

`patterns` - (optional) Match hardware design pins against the specified patterns.

Note: More than one pattern can be specified to find multiple pins based on different search criteria. You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets a list of pins attached to the specified cell:

```
hsi::get_pins -of [get_cells axi_gpio_0]
```

Note: If there are no pins matching the pattern, the tool will return a warning.

The following example gets a list of pins attached to the specified subsystem net:

```
hsi::get_pins -of [get_nets ps7_axi_interconnect_0_M_AXI_BRESP]
```

hsi::get_ports

Description

Get a list of external ports.

Syntax

```
get_ports [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet] [-
verbose] [<patterns>...]
```

Returns

Port objects. Returns nothing if the command fails.

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'port' objects of these types: 'hw_design bus_intf net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories

Hardware

Description

Gets a list of port objects in the current hardware design that match a specified search pattern. The default command gets a list of all ports in the hardware design.

The external connections in an hardware design are ports, or interface ports. The external connections in an IP integrator cell, or hierarchical module, are pins and interface pins. Use the `get_pins` and `get_intf_pins` commands to select the pin objects.

Arguments

`-regexp` – (optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and `-filter` expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `.*` to the beginning or end of a search string to widen the search to include a substring. See this web page for help with regular expression syntax.

Note: The Tcl built-in command `regexp` is not anchored, and works as a standard Tcl command. For more information, refer to this web page.

`-filter` <args> – (optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands.

Quote the filter search pattern to avoid having to escape special characters that might be found in net, pin, or cell names, or other properties. String matching is case sensitive and is always anchored to the start and to the end of the search string. The wildcard `*` character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the `*` wildcard character, this matches a property with a defined value of "".

For string comparison, the specific operators that can be used in filter expressions are `equal` (`==`), `not-equal` (`!=`), `match` (`=~`), and `not-match` (`!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by `AND` and `OR` (`&&` and `||`).

For IP subsystem ports, "DIRECTION", "TYPE", and "SENSITIVITY" are some of the properties you can use to filter results.

`-of_objects <arg>` - (optional) Get the ports connected to the specified IP subsystem nets returned by `get_nets`.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>.

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

`patterns` - (optional) Match ports against the specified patterns. The default pattern is the wildcard `*` which gets a list of all ports in the subsystem design. More than one pattern can be specified to find multiple ports based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples

The following example gets the ports connected to the specified hardware subsystem net:

```
hsi::get_ports -of_objects [get_nets bridge_1_apb_m] -filter {DIRECTION==I}
```

Note: If there are no ports matching the pattern, the tool will return a warning.

hsi::open_hw_design

Description

Open a hardware design from disk file.

Syntax

```
open_hw_design [-quiet] [-verbose] [<file>]
```

Returns

Hardware design object. Returns nothing if the command fails.

Usage

Name	Description
[`-quiet`]	Ignore command errors
[`-verbose`]	Suspend message limits during command execution
[<file>]	Hardware design file to open

Categories

Hardware

Description

Opens a Hardware design in the Hardware Software Interface. The hardware design must be exported previously using the Vivado product. Users can open multiple hardware designs at same time.

If successful, this command returns a hardware design object representing the opened Hardware design. Otherwise it returns an error.

Arguments

`-quiet` – (optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command line while launching the command are returned. Only errors occurring inside the command are trapped.

`-verbose` – (optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

file - The path and file name of the Hardware design to open in the HSM. The name must include the file extension.

Examples

Open the specified IP subsystem design in the current project:

```
open_hw_design C:/Data/project1/project1.sdk/SDK/SDK_Export/hw/design_1.xml
```

OR

```
open_hw_design  C:/Data/project1/project1.sdk/design_1_wrapper.xsa
```

Appendix A

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav, do the following:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **AMDDesignTools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs, do the following:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

Additional References

1. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
2. *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400)

Revision History

11/20/2025: Released with Vivado Design Suite 2025.2 without changes from 2025.1.

Section	Revision Summary
11/13/2024 Version 2024.2	
N/A	Editorial updates. No technical changes.
05/30/2024 Version 2024.1	
N/A	No updates.
12/13/2023 Version 2023.2	
N/A	Editorial changes only.
10/18/2023 Version 2023.2	
N/A	Updated for 2023.2 with editorial changes.
07/26/2023 Version 2023.1	
N/A	No changes to this section.
06/12/2023 Version 2023.1	
General updates	Editorial updates only. No technical content updates.
05/16/2023 Version 2023.1	
Memory and Register accesses from XSCT	Added topic.
Loading U-Boot over JTAG	Added topic for U-Boot flow using XSCT.
	Added support for including multiple dtsi files, instead of only one, in the device-tree.
01/02/2023 Version 2022.2	
General updates	Editorial updates only. No technical content updates.
12/23/2022 Version 2022.2	
N/A	No changes to this section.

Section	Revision Summary
10/19/2022 Version 2022.2	
N/A	• Fixed regression in platform config and platform list commands. • Supported template XSAs with createdts command. • Supported user dtsi files with createdts command. • Supported QEMU args files for Versal DC platforms. • Added new option to stapl config command to generate crc. • Cleared the entire TCM memory in Versal if any of the elf sections use tcm. • Added AI Engine to processors list returned by getprocessors command if the xsa contains AI Engine.
04/26/2022 Version 2022.1	
N/A	No changes to this section.

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright