Xilinx Software Command-Line Tool

Reference Guide

UG1208 (v2019.1) May 22, 2019

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/22/2019 Version 2019.1	
General Updates	This release is focused on quality. A number of quality related issues and bug fixes were addressed in this release.

Table of Contents

XILINX

Chapter 1

Xilinx Software Command-Line Tool

Graphical development environments such as the Xilinx® Software Development Kit (Xilinx SDK) are useful for getting up to speed on development for a new processor architecture. It helps to abstract away and group most of the common functions into logical wizards that even the novice can use. However, scriptability of a tool is also essential for providing the flexibility to extend what is done with that tool. It is particularly useful when developing regression tests that will be run nightly or running a set of commands that are used often by the developer.

Xilinx Software Command-line Tool (XSCT) is an interactive and scriptable command-line interface to Xilinx SDK. As with other Xilinx tools, the scripting language for XSCT is based on Tools Command Language (Tcl). You can run XSCT commands interactively or script the commands for automation. XSCT supports the following actions:

- Create hardware, board support packages (BSPs), and application projects
- Manage repositories
- Set toolchain preferences
- Configure and build BSPs/applications
- Download and run applications on hardware targets
- Create and flash boot images by running Bootgen and program_flash tools.

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

As you read the document you will notice usage of some abbreviations for various products produced by Xilinx. For example:

- Use of `ps7` in the source code implies that these files are targeting the 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 a 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 (HDF) are used to transfer the information about the hardware system that includes a processor to the embedded software development tools such as Xilinx SDK (XSDK) and Xilinx Software Command-Line Tools (XSCT). It includes information about which peripherals are instantiated, clocks, memory interfaces, and memory maps.

- Microprocessor Software Specification (MSS) files are used to store information about the BSP. They contain OS information for the BSP, software drivers associated with each peripheral of the hardware design, STDIO settings, and compiler flags like optimization and debug information level.

System Requirements

If you plan to use capabilities that are offered through the Xilinx® SDK or the Xilinx Software Command-Line Tool (XSCT), then you also need to meet the hardware and software requirements that are specific to that capability.

Hardware Requirements

The table below lists the hardware requirements.

Table 1: **Hardware Requirements**

Requirement	Description
CPU Speed	2.2 GHz minimum or higher; Hyper-threading (HHT) or multicore recommended.
Processor	Intel Pentium 4, Intel Core Duo, or Xeon Processors; SSE2 minimum
Memory/RAM	2 GB or higher
Display Resolution	1024×768 or higher at normal size (96 dpi)
Disk Space	Based on the components selected during the installation

Software Requirements

The table below lists the supported operating systems.

***Note*:** 32-bit machine support is now only available through Lab Edition and Hardware Server standalone product installers.

Table 2: **Software Requirements**

Operating System	Supported Version
Windows	• Windows 7 SP1 (64-bit) • Windows 8.1 (64-bit) • Windows 10 Pro (64-bit)

Table 2: **Software Requirements** *(cont'd)*

Operating System	Supported Version
Linux	• Red Hat Enterprise Linux: ◦ 6.6-6.9 (64-bit) ◦ 7.0-7.1 (64-bit) • CentOS: ◦ 6.7-6.8 (64-bit) ◦ 7.2-7.3 (64-bit) • SUSE Linux Enterprise: ◦ 11.4 (64-bit) ◦ 12.2 (64-bit) • Ubuntu Linux 16.04.2 LTS (64-bit) ***Note*:** Additional library installation required.

Installing and Launching XSCT

The Xilinx® Software Command-Line Tool (XSCT) can be installed either as a part of the Xilinx SDK installer or as a separate command-line tool only installation. XSCT is available for the following platforms:

- Microsoft Windows
- Linux

The following sections explain the installation process for each of these platforms.

Installing and Launching XSCT on Windows

XSCT can be installed using the Windows executable installer. The installer executable bears the name `Xilinx_SDK_<version>_Win64.EXE`, where `<version>` indicates the Xilinx Software Development Kit (Xilinx SDK) version number.

Note: Installing XSCT on Microsoft Windows operating system might require administrator rights. In addition, your project workspace needs to be set up in any folder that you can fully access.

1. To install XSCT, double-click the Windows installer executable file.
2. The installer accepts your login credentials and allows you to select specific tool components. The client then automatically downloads only what you have selected and installs it on your local machine.
3. In the Select Edition to Install window, select the **Xilinx Software Command-Line Tool (XSCT)** option to install XSCT as a seperate command-line tool only. Alternatively, you can also select the **Xilinx Software Development Kit (XSDK)** option to install XSCT as a part of the Xilinx SDK, an Eclipse-based integrated development environment.
4. Unless you choose otherwise, XSCT is installed in the `C:\Xilinx` directory.
5. To launch XSCT on Windows, select **Start → Programs → Xilinx Design Tools → SDK <version>** and then select **Xilinx Software Command Line Tool**. Where **SDK <version>** indicates the Xilinx Software Development Kit version number.
6. You can also launch XSCT from the command line.

```
cd C:\Xilinx\SDK\<version>\bin
                xsct.bat
```

7. To view the available command-line options, issue the `help` command at the XSCT command prompt.

```
****** Xilinx Software Commandline Tool (XSCT)

                  ** Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.

                  xsct% help
                  Available Help Categories

                  connections    - Target Connection Management
                  registers      - Target Registers
                  running        - Program Execution
                  memory         - Target Memory
                  download       - Target Download FPGA/BINARY
                  reset          - Target Reset
                  breakpoints    - Target Breakpoints/Watchpoints
                  streams        - Jtag UART
                  miscellaneous  - Miscellaneous
                  jtag           - JTAG Access
                  sdk            - SDK Projects
                  petalinux      - Petalinux commands
                  hsi            - HSI commands

                  Type "help" followed by above "category" for more
details or
                  help" followed by the keyword "commands" to list all
the commands

                  xsct%
```

Installing and Launching XSCT on Linux

Xilinx Software Command-line Tool (XSCT) can be installed using the small self-extracting web install executable binary distribution file. The installer file bears the name `Xilinx_SDK_<version>_Lin64.BIN`, where `<version>` indicates the Xilinx Software Development Kit (Xilinx SDK) version number.

***Note*:** The procedure for installing XSCT on Linux depends on which Linux distribution you are using. Ensure that the installation folder has the appropriate permissions. In addition, your project workspace needs to be set up in any folder that you can fully access.

1. To install XSCT, launch the terminal and change the permission of the self-extracting binary executable.

```
$ chmod +x Xilinx_SDK_<version>_Lin64.BIN
```

2. Start the installation process or run the `.BIN` file.

```
./Xilinx_SDK_<version>_Lin64.BIN
```

3. The installer accepts your login credentials and allows you to select specific tool components. The client then automatically downloads only what you have selected and installs it on your local machine.
4. In the Select Edition to Install window, select the **Xilinx Software Command-Line Tool (XSCT)** option to install XSCT as a seperate command-line tool only. Alternatively, you can also select the **Xilinx Software Development Kit (XSDK)** option to install XSCT as a part of the Xilinx SDK, an Eclipse-based integrated development environment.
5. Unless you choose otherwise, XSCT is installed in the `/opt/Xilinx` directory.
6. To launch XSCT on Linux, select **Applications → Other** and then select **Xilinx Software Command Line Tool <version>**. Where **<version>** is the version number of the XSCT.
7. You can also launch XSCT from the command line.

```
cd /opt/Xilinx/SDK/<version>/bin
./xsct
```

8. To view the available command-line options, issue the `help` command at the XSCT command prompt.

```
****** Xilinx Software Commandline Tool (XSCT)

    ** Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.

xsct% help
Available Help Categories

connections    - Target Connection Management
registers      - Target Registers
running        - Program Execution
memory         - Target Memory
download       - Target Download FPGA/BINARY
reset          - Target Reset
breakpoints    - Target Breakpoints/Watchpoints
streams        - Jtag UART
miscellaneous  - Miscellaneous
jtag           - JTAG Access
sdk            - SDK Projects
petalinux      - Petalinux commands
hsi            - HSI commands

Type "help" followed by above "category" for more details or
help" followed by the keyword "commands" to list all the commands

xsct%
```

XSCT Commands

The Xilinx® Software Command-Line tool allows you to create complete Xilinx SDK workspaces, investigate the hardware and software, debug and run the project, all from the command line.

XSCT commands are broadly classified into the following categories. The commands in each category are described subsequently.

- Target Connection Management
- Target Registers
- Program Execution
- Target Memory
- Target Download FPGA/BINARY
- Target Reset
- Target Breakpoints/Watchpoints
- Jtag UART
- Miscellaneous
- JTAG Access
- SDK Projects
- HSI Commands

TIP:

- Help for each of the commands can be viewed by running `help <command>` or `<command> -help` in the XSCT console. All the available XSCT commands can be listed by running `help commands`.
- You can use **Ctrl+C** to terminate long running commands like `fpga` or `elf download` or `for/while` loops.
- You can terminate XSCT by pressing **Ctrl+C** twice in succession.
- Windows style paths are supported when the path is enclosed within curly brackets `{}`.

Target Connection Management

The following is a list of connections commands:

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 exist to the same url
`-xvc-url <url>`	Open Xilinx Virtual Cable connection

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

Example(s)

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

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

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

Identical to 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 comibination with -filter option. An error will be generate if list is empty or contains more than one entry.

Option	Description
`-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 references by name, while Tcl variables are accessed using the $ syntax, string must be quoted. Operators ==, !=, `<=`, >=, `<`, >, `&&` and `||` are supported as well as `()`. These 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 dict's containing target properties.
`-index <index>`	Include targets based on 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. The timeout value is in seconds. Default timeout is 3 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.

Example(s)

```
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 1st Jtag Device.

gdbremote connect

Connect to GDB remote server.

Syntax

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

Connect to a GDB remote server, for example qemu. A special client named tcfgdbclient is used to connect to remote GDB server.

Options

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

Returns

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

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, 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.

Example(s)

```
rrd
```

Read top level registers or groups.

```
rrd r0
```

Read register r0.

```
rrd usr r8
```

Read register r8 in group usr.

```
rrd sys 5
```

ESR_ELn registers are available in sys group of registers.

```
rrd sys 6
```

FAR_ELn registers are available through sys 6.

rwr

Write to register

Syntax

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

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

Returns

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

Example(s)

```
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

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.

Example(s)

```
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 5 sec 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 instruction is function call, stop at 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 instruction is 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 current PC Default value for `<num>` is 1.

Returns

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

Example(s)

```
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 a complex data types like a structure can be accessed through '.' operator. For example, var1.int_type refers to int_type element in 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 auto expression list. The values or definitions of the expressions in auto expression list are displayed when 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, 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.

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

`-defs`: Expression definition(s)

`-remove or -set`: Nothing

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

Example(s)

```
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 variable name and value are not specified, values of all the local variables are returned. Elements of a complex data types like a structure can be accessed through '.' operator. For example, var1.int_type refers to int_type element in 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.
`-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.

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.

Example(s)

```
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 int_type element in var1 struct, where var1 is a member of 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 var1 struct, which is in turn a member of tmp_var struct.

backtrace

Stack back trace.

Syntax

```
backtrace
```

Return stack trace for current target. Target must be stopped. Use debug information for best result.

Returns

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

profile

Configure and run the GNU profiler.

Syntax

```
profile [options]
```

Configure and run the GNU profiler. The profiling needs to enabled while building 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, as 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 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.

Example(s)

```
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 MB. The output file is generated in gmon.out format. The results can be viewed using gprof editor. In case of cycle count, an annotated disassembly file is also generated clearly marking 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. Default is 100MHz.
`-count-instr`	Count no. of executed instructions. By default no. 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 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.

Example(s)

```
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 no. of instructions executed.

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 will be aligned to access-size before reading memory, if '-unaligned-access' option is not used. For targets which do not support double-word access, debugger uses 2 word accesses. If number of data values to be read is more than 1, then debugger selects appropriate access size. For example, 1. mrd -size b 0x0 4 Debugger accesses one word from the memory, displays 4 bytes. 2. mrd -size b 0x0 3 Debugger accesses one half-word and one byte from the memory, displays 3 bytes. 3. mrd 0x0 3 Debugger accesses 3 words from the memory and displays 3 words.
`-value`	Return a Tcl list of values, instead of displaying the result on 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 -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 8 bits select an AP and lower 8 bits are the register address for that AP.
`-unaligned-access`	Memory address is not aligned to 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 a 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.

Example(s)

```
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 3 bytes at address 0x1.

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

Read 2 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 8 bits (0x1) select the APB-AP and lower 8 bits (0x0) is the address of CSW.

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

Read AHB-AP TAR on Zynq. The higher 8 bits (0x0) select the AHB-AP and lower 8 bits (0x4) is the address of TAR.

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

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

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

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

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 specifed 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 specifed by <address>.

Options

Option	Description
-force	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked.
-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 will be aligned to accesss-size before writing to memory, if '-unaligned-access' option is not used. If target does not support double-word access, the debugger uses 2 word accesses. If number of data values to be written is more than 1, then debugger selects appropriate access size. For example, 1. mwr -size b 0x0 {0x0 0x13 0x45 0x56} Debugger writes one word to the memory, combining 4 bytes. 2. mwr -size b 0x0 {0x0 0x13 0x45} Debugger writes one half-word and one byte to the memory, combining the 3 bytes. 3. mwr 0x0 {0x0 0x13 0x45} Debugger writes 3 words to the memory.
-bin	Read binary data from a file and write it to target address space.
-file <file-name>	File from which binary data is read to write to 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 -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 8 bits select an AP and lower 8 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 a APU target to access ARM DAP and MEM-AP address space.

Returns

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

Example(s)

```
mwr 0x0 0x1234
```

Write 0x1234 to address 0x0.

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

Write 4 words from the list of values to address 0x0.

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

Write 4 words from the list of values to address 0x0 and fill the last word from the list at remaining 6 address locations.

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

write 3 bytes from the list at address 0x1.

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

write 2 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 8 bits (0x1) select the APB-AP and lower 8 bits (0x0) is the address of CSW.

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

Write 0xf8000120 to AHB-AP TAR on Zynq The higher 8 bits (0x0) select the AHB-AP and lower 8 bits (0x4) is the address of TAR.

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

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

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

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

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 specifed and only one symbol file exists in target's memory map, then that symbol file is used. If no symbol file is specifed and multiple symbol files exist in target's memory map, then 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 will not be tracked for debug and are not visible in the debug targets view.
`-fast-step`	Enable fast stepping. Only the current process will be re-synced after stepping. All other processes will not be re-synced when this flag is turned on.

Note(s)

- fast-exec and fast-step options are not valid with disable option.

Returns

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

Example(s)

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

Enable OSA for `<symbole-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.
`-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 memmap command can be removed.

Returns

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

Example(s)

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

Add the memory region 0xfc000000 - 0xfc000fff to 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 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).
`-keepsym`	Keep previously downloaded elfs in the list of symbol files. Default behavior is to clear the old symbol files while downloading an elf.
`-force`	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked.
`-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 info for the code. `<addr>` is the relative address, to which all the program sections are relocated.

Returns

Nothing.

verify

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

Syntax

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

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

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

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

Options

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

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 current configuration. This options should be used while configuring partial bitstreams created before 2014.3 or any partial bitstreams in binary format.

Option	Description
`-no-revision-check`	Disable bitstream vs silicon revision revision compatibility check.
`-skip-compatibility-check`	Disable bitstream vs FPGA device 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 device, and only one supported FPGA device is found in the targets list, then this device will be 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 Configutation 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. A processor group is defined as a set of processors and on-chip peripherals like OCM.
`-system`	Reset the active System.
`-srst`	Generate system reset for active target. With JTAG this is done by generating a pulse on the SRST pin on the JTAG cable assocated with the active target.

Returns

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

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 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 hw_server/TCF agent. This is the default type hw = Hardware Breakpoint sw = Software Breakpoint
`-mode <breakpoint-mode>`	Specify the access mode that will trigger the breakpoint. `<breakpoint-mode>` can be a bitwise OR of the values below: 0x1 = Triggered by a read from the breakpoint location 0x2 = Triggered by a write to the breakpoint location 0x4 = Triggered by an instruction execution at the breakpoint location This is the default for Line and Address breakpoints 0x8 = 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.

Option	Description
`-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.
`-properties <dict>`	Specify advanced breakpoint properties.
`-meta-data <dict>`	Specify meta-data 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, then the breakpoint is set for the active target selected through targets command. If there is no active target, then the breakpoint is set for all targets.
`-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 = supress trigger on stepping over a code breakpoint 2 = supress trigger on any kind of stepping

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 will be enabled only for that active target. If there is no active target, the Breakpoint will be enabled for all the targets. 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.

Example(s)

```
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 \"-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.

Example(s)

```
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.

Example(s)

```
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 \"-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.

Example(s)

```
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 Beakpoints/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 Breakpoint hitcount 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 hyper-terminal.

Syntax

```
jtagterminal [options]
```

Start/Stop a JTAG based hyper-terminal to communicate with 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 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. 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 a MDM or ARM processor target before runnning this command.
- While running a script in non-interactive mode, output from Jtag uart may not 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.

Example(s)

```
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 HW design.

Syntax

```
loadhw [options]
```

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

Options

Option	Description
`-hw`	HW design file.
`-list`	Return a list of open designs for the targets.
`-mem-ranges [list {start1 end1} {start2 end2}]`	List of memory ranges from which the memory map should be set. Memory map is not set for the addresses outside these ranges. If this option is not specified, then memory map is set for all the addresses in the hardware design.

Returns

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

Example(s)

targets -filter {name =~ "APU"}; loadhw design.hdf Load the HW design named design.hdf and set memory map for all the child processors of APU target. targets -filter {name =~ "xc7z045"}; loadhw design.hdf Load the HW design named design.hdf and set memory map for all the child processors for which xc7z045 is the parent.

unloadhw

Unload a Vivado HW design.

Syntax

```
unloadhw
```

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

Returns

Nothing.

mdm_drwr

Write to MDM Debug Register.

Syntax

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

Write to MDM Debug Register. cmd is 8-bit MDM command to access a Debug Register. data is the register value and bitlen is the register width.

Options

Option	Description
`-user <bscan number>`	Specify user bscan port number. Default is 2.

Returns

Nothing, if successful. Error string, if BSCAN port is invalid.

Example(s)

```
mdm_drwr 8 0x40 8
```

Write to MDM Break/Reset Control Reg.

mb_drwr

Write to MicroBlaze Debug Register.

Syntax

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

Write to MicroBlaze Debug Register available on MDM. cmd is 8-bit MDM command to access a Debug Register. data is the register value and bitlen is the register width.

Options

Option	Description
`-user <bscan number>`	Specify user bscan port number. Default is 2.

Returns

Nothing, if successful. Error string, if BSCAN port is invalid.

Example(s)

```
mb_drwr 1 0x282 10
```

Write to MB Control Reg.

mdm_drrd

Read from MDM Debug Register.

Syntax

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

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

Options

Option	Description
`-user <bscan number>`	Specify user bscan port number. Default is 2.

Returns

Register value, if successful. Error string, if BSCAN port is invalid.

Example(s)

```
mdm_drrd 0 32
```

Read XMDC ID Reg.

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
`-user <bscan number>`	Specify user bscan port number. Default is 2.

Returns

Register value, if successful. Error string, if BSCAN port is invalid.

Example(s)

```
mb_drrd 3 28
```

Read MB Status Reg.

configparams

List, get or set configuration parameters.

Syntax

```
configparams <options>
```

List 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 may change depending on 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 represet the global default. Not all options support context specific values.

Returns

Depends on the arguments specified.

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

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

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

Example(s)

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

Disable access protection for dow, mrd, and mwr commands.

```
configparams sdk-launch-timeout 100
```

Change the SDK launch timeout to 100 seconds, used for running SDK batch mode commands.

version

Get SDK or TCF server version.

Syntax

```
version [options]
```

Get SDK or TCF server version. When no option is specified, SDK build version is returned.

Options

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

Returns

SDK or TCF Server version, on success. Error string, if server verison is requested when there is no connection.

xsdbserver start

Start XSDB command server.

Syntax

```
xsdbserver start [options]
```

Start XSDB command server listener. 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 the 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 incomming connections.
`-port <port>`	Specifies port to listen on. If this option is not specified or if the port is zero then a dynamically allocated port number is used.

Returns

Server details are disaplayed on the console if server is started. successfully, or error string, if a server has been already started.

Example(s)

```
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 incomming 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

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 comibination with -filter option. An error will be generate if list is empty or contains more than one entry.
`-regexp`	Use regexp for filter matching.
`-nocase`	Use case insensitive filter matching.

Option	Description
`-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 references by name, while Tcl variables are accessed using the $ syntax, string must be quoted. Operators ==, !=, <=, >=, <, >, && and || are supported as well as (). 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 3 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.

Example(s)

```
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 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.

Example(s)

```
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 will ensure 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 since 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 will attept 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 will return 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.

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 enumberate or connect to available JTAG ports.

Options

Option	Description
`-list`	List opened servers. This is the default if no other option is given.
`-format`	List format of supported server strings.
`-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.

Example(s)

```
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

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 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.

Option	Description
`-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 used to specify the chunk size in bytes for each transaction while downloading. Supported only for Microblaze processors.

Returns

Nothing

Example(s)

```
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.

Example(s)

```
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.

Example(s)

```
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.

Example(s)

```
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

Example(s)

```
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

Example(s)

```
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

Example(s)

```
svf delay 1000
```

Delay of 1000 tcks is added to the SVF file.

SDK Projects

The following is a list of sdk commands:

- openhw
- closehw
- openbsp
- closebsp
- updatemss
- getaddrmap
- getperipherals
- repo
- configbsp
- setlib
- removelib
- getlibs
- setdriver
- getdrivers
- setosversion
- getos
- regenbsp
- setws
- getws
- createhw
- updatehw
- changebsp
- createbsp
- createapp
- createlib
- projects
- importprojects

- importsources
- getprojects
- deleteprojects
- configapp
- toolchain

openhw

Open a hardware design.

Syntax

```
openhw <hw-proj | hdf/xml file>
```

Open a hardware design exported from Vivado. HDF/XML file 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.

Example(s)

```
openhw ZC702_hw_platform
```

Open the hardware project ZC702_hw_platform.

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

Open the hardware project corresponding to the system.hdf.

closehw

Close a hardware design.

Syntax

```
closehw <hw project | hdf/xml file>
```

Close a hardware design that was opened using 'openhw' command. HDF/XML file 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.

Example(s)

```
closehw ZC702_hw_platform
```

Close the hardware project ZC702_hw_platform.

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

Close the hardware project corresponding to the system.hdf.

openbsp

Open the BSP.

Syntax

```
openbsp <bsp-proj | mss-file>
```

Open the BSP from BSP project created using 'createbsp', or from the MSS file.

Options

None

Returns

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

Example(s)

```
openbsp hello_bsp
```

Open the BSP project 'hello_bsp'.

```
openbsp /tmp/wrk/hello_bsp/system.mss
```

Open the bsp project corresponding to the system.mss.

closebsp

Close the BSP.

Syntax

```
closebsp <bsp-proj | mss-file>
```

Close the BSP project specified by `<bsp-project>` or the BSP project corresponding to the MSS file specified by `<mss-project>`.

Options

None

Returns

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

Example(s)

```
closebsp hello_bsp
```

Close the BSP project 'hello_bsp'.

```
closebsp /tmp/wrk/hello_bsp/system.mss
```

Close the BSP project corresponding to the system.mss.

updatemss

Update the mss file with the changes done to the BSP.

Syntax

```
updatemss [OPTIONS]
```

Update the mss file with changes done to the BSP.

Options

Option	Description
`-mss <mss file>`	MSS file to be updated.

Returns

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

Example(s)

```
updatemss -mss system.mss
```

Update system.mss file with the changes done to the BSP.

getaddrmap

Get the address ranges of IP connected to processor.

Syntax

```
getaddrmap <hw proj | 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.

Example(s)

```
getaddrmap hw1 ps7_cortexa9_0
```

Return the address map of peripherals connected to ps7_cortexa9_0. hw1 is the hw project, which is created using command 'createhw'.

```
getaddrmap system.hdf ps7_cortexa9_0
```

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

getperipherals

Get a list of all peripherals in the HW design.

Syntax

```
getperipherals <hw proj | hdf/xml file> <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.

Example(s)

```
getperipherals system.hdf
```

Return a list of peripherals in the hardware design.

```
getperipherals system.hdf ps7_cortexa9_0
```

Return a list of peripherals connected to processor CortexA9#0 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 the repositories.

Returns

Depends on the OPTIONS specified.

`-scan, -set <path-list>`: Returns nothing.

`-get`: Returns the current repository path.

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

Example(s)

```
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.

configbsp

Configure settings for BSP projects.

Syntax

```
configbsp [OPTIONS] [<param-name> [<value>]]
```

If `<param-name>` and `<value>` are not specified, returns the details of all configurable parameters of processor, os, or all libraries in BSP. If `<param-name>` is specified and `<value>` value is not specified, return the value of the parameter. If `<param-name>` and `<value>` are specified, set the value of parameter.

Options

Option	Description
`-bsp <bsp-proj | mss file>`	BSP project or mss file.
`-proc`	Return the configurable parameters of processor in BSP.
`-os`	Return the configurable parameters of OS in BSP.
`-lib <lib-name>`	Return the configurable parameters of library `<lib-name>` in BSP.
`-append`	Append the value to the parameter in BSP.

Returns

Depends on the arguments specified.

`<none>`: List of parameters and description of each parameter of processor, os, or libs depending on the option specified.

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

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

Example(s)

```
configbsp -bsp system.mss -os
```

Return the list of configurable parameters of the OS in the BSP.

```
configbsp -bsp system.mss -proc
```

Return the list of configurable parameters of processor in the BSP.

```
configbsp -bsp bsp1 -lib xilisf
```

Return the list of configurable parameters of library "xilisf" in the BSP.

```
configbsp -bsp system.mss extra_compiler_flags
```

Return the value of parameter 'extra_compiler_flags' in the BSP.

```
configbsp -bsp system.mss extra_compiler_flags "-pg"
```

Set "-pg" as the value of parameter 'extra_compiler_flags' in the BSP.

```
configbsp -bsp system.mss -append extra_compiler_flags "-pg"
```

Append "-pg" to the value of parameter 'extra_compiler_flags' in the BSP

setlib

Set library and version in BSP.

Syntax

```
setlib [OPTIONS]
```

Add a library to BSP. If version is not specified, latest library version available is added. If library is already available in BSP, the library version is updated.

Options

Option	Description
`-bsp <bsp-proj | mss file>`	BSP project or mss file.
`-lib <lib-name>`	Library name to be added to BSP.
`-ver <lib-ver>`	Library version to be added. Default latest version of library is added.

Returns

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

Example(s)

```
setlib -bsp hello_bsp -lib xilffs -ver 2.0
```

Add xilffs library version 2.0 to the BSP.

```
setlib -bsp hello_bsp -lib xilrsa
```

Add latest version of xilrsa library available in repo, to the BSP.

removelib

Remove library from BSP.

Syntax

```
removelib [OPTIONS]
```

Remove specified library from BSP.

Options

Option	Description
`-bsp <bsp-proj | mss file>`	BSP project or mss file.
`-lib <lib-name>`	Library name to be removed from BSP.

Returns

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

Example(s)

```
removelib -bsp hello_bsp -lib xilffs
```

Remove xilffs library from the BSP.

getlibs

Get libraries from BSP.

Syntax

```
getlibs [OPTIONS]
```

Return a list of libraries and their versions from BSP in tabular format.

Options

Option	Description
`-bsp <bsp-proj | mss-file>`	BSP project or mss file.

Returns

If successful, this command returns the library details. Otherwise it returns an error.

Example(s)

```
getlibs -bsp hello_bsp
```

Return the list of all libraries in the BSP.

setdriver

Set driver for IP in the BSP.

Syntax

```
setdriver [OPTIONS]
```

Set specified driver to the IP core in the BSP.

Options

Option	Description
`-bsp <bsp-proj | mss-file>`	BSP project or mss file to which the driver needs to be added.
`-ip <ip-name>`	IP instance for which driver needs to be set.
`-driver <driver-name>`	Driver name which needs to be added to the BSP.
`-ver <driver-version>`	Version of the driver.

Returns

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

Example(s)

```
setdriver -bsp hello_bsp -ip ps7_uart -driver generic -ver 2.0
```

Set the generic driver for the ps7_uart IP in the BSP.

getdrivers

Get drivers from BSP.

Syntax

```
getdrivers [OPTIONS]
```

Return the list of drivers from the BSP in tabular form.

Options

Option	Description
`-bsp <bsp-proj | mss-file>`	BSP project or mss file.

Returns

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

Example(s)

```
getdrivers -bsp hello_bsp
```

Return the list of drivers assigned to the IPs in the BSP.

setosversion

Set OS version in BSP.

Syntax

```
setosversion [OPTIONS]
```

Set specified OS version in the BSP. Latest version is added by default.

Options

Option	Description
`-bsp <bsp-proj | mss-file>`	BSP project or mss file corresponding to the OS.
`-ver <os-version>`	Version of the OS. Default latest version of OS is added.

Returns

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

Example(s)

```
setosversion -bsp hello_bsp -ver 5.4
```

Set the OS version 5.4 in the BSP.

```
setosversion -bsp hello_bsp
```

Set the latest OS version from repo in the BSP.

getos

Get OS details from BSP.

Syntax

```
getos [OPTIONS]
```

Return OS details from the BSP.

Options

Option	Description
`-bsp <bsp-proj | mss-file>`	BSP project or mss file corresponding to the OS.

Returns

If successful, this command returns OS details, otherwise it returns an error.

Example(s)

```
getos -bsp hello_bsp
```

Return the OS details in the BSP.

regenbsp

Regenerate BSP sources.

Syntax

```
regenbsp [OPTIONS]
```

Regenerate the sources with the modifications made to BSP.

Options

Option	Description
`-bsp <bsp-proj | mss-file>`	BSP project or mss file.

Returns

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

Example(s)

```
regenbsp -bsp hello_bsp
```

Regenerate the BSP sources with the changes done in the BSP settings.

setws

Set SDK workspace

Syntax

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

Set SDK 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.

Example(s)

```
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 SDK workspace

Syntax

```
getws
```

Return the current SDK workspace.

Returns

Current workspace.

createhw

Create a hardware project.

Syntax

```
createhw [OPTIONS]
```

Create a hardware project using a hardware specification file.

Options

Option	Description
`-name <project-name>`	Project name that should be created.
`-hwspec <HW specfiction file>`	Hardware specification file for creating a hardware project.

Returns

Nothing, if the hardware project is created successfully. Error string, if invalid options are used or if the project cannot be created.

Example(s)

```
createhw -name hw1 -hwspec system.hdf
```

Create a hardware project with name hw1 from the hardware specification file system.hdf.

updatehw

Update a hardware project.

Syntax

```
updatehw [OPTIONS]
```

Update a hardware project with the changes in the new hardware specification file.

Options

Option	Description
`-hw <hw-project>`	Hardware project that should be updated.
`-newhwspec <hw specfiction file>`	New hardware specification file for updating the hardware project.

Returns

Nothing, if the HW project is updated successfully. Error string, if invalid options are used or if the project cannot be updated.

Example(s)

```
updatehw -hw hw1 -newhwspec system.hdf
```

Update the hardware project hw1 with the changes in the new hardware specification file system.hdf.

changebsp

Change the referenced BSP for an application.

Syntax

```
changebsp [OPTIONS]
```

Change the referenced BSP for an application project.

Options

Option	Description
`-app <application-project>`	Application project whose BSP should be changed.
`-newbsp <BSP Project>`	New BSP which the application should refer.

Returns

Nothing, if the Application Project refernece changed. Error string, if invalid options are used or if the project cannot be updated.

Example(s)

```
changebsp -app app1
```

Lists the bsp projects which can be referenced by the given application project changebsp -app app1 -newbsp test2_bsp. Change the referneced bsp for app1 to test2_bsp.

createbsp

Create a bsp project

Syntax

```
createbsp [OPTIONS]
```

Create a bsp project for the specified hardware and processor.

Options

Option	Description
`-name <project-name>`	Project name that should be created
`-proc <processor-name>`	Processor instance that should be used for creating bsp project.
`-hwproject <hw project name>`	Hardware project for which the application or bsp project should be created.
`-os <OS name>`	OS type for the application project. Default type is standalone
`-mss <MSS File path>`	MSS File path for creating BSP. This option can be used for creating a BSP from user mss file. When mss file is specified, then processor and os options will be ignored and processor/os details are extracted from mss file.
`-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. This is valid only for A53 processors, defaults to 32-bit for other processors.

Returns

Nothing, if the BSP project is created successfully. Error string, if invalid options are used or if the project cannot be created.

Example(s)

```
createbsp -name bsp1 -hwproject hw1 -proc ps7_cortexa9_0
```

Create a BSP project with name bsp1 from the hardware project hw1 for processor 'ps7_cortexa9_0'

```
createbsp -name bsp1 -hwproject hw1 -proc ps7_cortexa9_0 -os standalone
```

Create a BSP project with name bsp1 from the hardware project hw1 for processor 'ps7_cortexa9_0' and standalone OS.

```
createbsp -name bsp1 -hwproject hw1 -mss system.mss
```

Create a BSP project with name bsp1 with all the details from system.mss

```
createbsp -name bsp1 -hwproject hw1 -proc psu_cortexa53_0 -arch 32
```

Create a BSP project with name bsp1 for psu_cortexa53_0 using a 32 bit tool chain.

createapp

Create an application project.

Syntax

```
createapp [OPTIONS]
```

Create an application project from a list of template applications.

Options

Option	Description
`-name <project-name>`	Project name that should be created.
`-app <template-application-name>`	Name of the template application. Default is "Hello World". Use 'repo -apps' command to get the list of all application templates.
`-proc <processor-name>`	Processor instance that should be used for creating application project.
`-hwproject <hw project name>`	Hardware project for which the application or bsp project should be created.
`-bsp <BSP project name>`	BSP project for which the application project should be created. If this option is not specified, a default bsp is created
`-os <OS name>`	OS type for the application project. 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. This is valid only for A53 processors, defaults to 32-bit for other processors.

Returns

Nothing, if the Application project is created successfully. Error string, if invalid options are used or if the project cannot be created.

Example(s)

```
createapp -name hello1 -bsp bsp1 -hwproject hw1 -proc ps7_cortexa9_0
```

Create a default application "Hello World" project with name 'hello1' for processor 'ps7_cortexa9_0'.

```
createapp -name fsbl1 -app {Zynq FSBL} -hwproject hw1 -proc ps7_cortexa9_0
```

Create a Zynq FSBL project with name 'fsbl1' and also creates a BSP 'fsbl1_bsp' for processor 'ps7_cortexa9_0' and default OS 'standalone'.

```
createapp -name e1 -app {Empty Application} -hwproject hw2
```

-proc microblaze_0 -lang c++ Create an empty C++ application project with name 'e1'.

```
createapp -name hello2 -app {Hello World} -hwproject hw1
```

-proc psu_cortexa53_0 -arch 32 Create a Hello World application project with name 'hello2' for processor 'psu_cortexa53_0' with 32-bit tool chain.

createlib

Create a library project.

Syntax

```
createlib [OPTIONS]
```

Create static or shared library project.

Options

Option	Description
`-name <project-name>`	Project name that should be created.
`-type <library-type>`	`<library-type>` can be 'static' or 'shared' Default type is 'shared'.
`-proc <processor-type>`	Processor type that should be used for creating application project. 'ps7_cortex9', 'microblaze', 'psu_cortexa53' or 'psu_cortexr5'.
`-os <OS name>`	OS type for the application project. 'linux' or 'standalone' Default type is linux.
`-lang <programming-language>`	`<programming-language>` can be 'c' or 'c++'. Default is c.
`-arch <arch-type>`	Processor architecture, `<arch-type>` can be 32 or 64 This option is used to build the project with 32/64 bit toolchain. This is valid only for A53 processors, defaults to 32-bit for other processors.
`-flags <compiler-flags>`	Optional - compiler flags.

Returns

Nothing, if the library project is created successfully Error string, if invalid options are used or if the project cannot be created.

Example(s)

```
createlib -name lib1 -type static -proc ps7_cortexa9
```

Create a static library project with name 'lib1' for processor 'ps7_cortexa9' and default os 'standalone' with default language 'C'.

```
createlib -name lib2 type shared -proc psu_cortexa53 -os linux -lang C++
```

Create a shared library project with name 'lib2' for processor 'psu_cortexa53' and Linux OS with C++ language.

```
createlib -name st-stnd-r5-c-flags -type static -proc psu_cortexr5
```

-os standalone -lang C -arch 32 -flags {-g3 -pg} Create a static library project with name 'st-stnd-r5-c-flags' for processor 'psu_cortexr5' and standalone OS with C language with extra compiler flags '-g3 -pg'.

projects

Build/Clean projects.

Syntax

```
projects [OPTIONS]
```

Build/Clean a bsp/application project or all projects in workspace.

Options

Option	Description
`-build | -clean`	Build / Clean projects.
`-type <project-type>`	`<project-type>` can be "all", "bsp" or "app" Default type is all.
`-name <project-name>`	Name of the project that should be built.

Returns

Nothing, if the project is built successfully. Error string, if invalid options are used or if the project cannot be built.

Example(s)

```
projects -build -type bsp -name hello_bsp
```

Build the BSP project 'hello_bsp'.

```
projects -build
```

Build all the projects in the current workspace.

```
projects -clean -type app
```

Clean all the application projects in the current workspace.

```
projects -clean
```

Clean all the projects in the current workspace.

importprojects

Import projects to workspace.

Syntax

```
importprojects <path>
```

Import all the SDK 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.

Example(s)

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

Import sdk 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.
`-path <source-path>`	Path from which the source files should be imported. All the files/directories from the `<source-path>` are imported to application project. All existing source files will be overwritten in the application, and new ones will be copied. Linker script will not be copied to the application 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 imported.

Example(s)

```
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.

getprojects

Get projects from the workspace.

Syntax

```
getprojects [OPTIONS]
```

Get hw/bsp/application projects or all projects from the workspace.

Options

Option	Description
`-type <project-type>`	`<project-type>` can be "all", "hw", "bsp" or "app" Default type is all.

Returns

List of all the projects of type `<project-type>` in the workspace.

Example(s)

```
getprojects
```

Return the list of all the projects available in the current workspace.

```
getprojects -type hw
```

Return the list of hardware projects.

deleteprojects

Delete project(s) from the workspace.

Syntax

```
deleteprojects [OPTIONS]
```

Delete project(s) from the workspace or disk.

Options

Option	Description
`-name`	Project name/list to be deleted List of projects should be separated by semi-colon {proj1;proj2;proj3}.
`-workspace-only`	Delete project from workspace only and not from disk. Default operation is to delete projects from disk.

Returns

Nothing, if the projects are deleted successfully. Error string, if invalid options are used or if the project cannot be deleted.

Example(s)

```
deleteprojects -name hello1
```

Delete the hello1 project from the disk.

```
deleteprojects -name hello1 -workspace-only
```

Delete the hello1 project from workspace only.

configapp

Configure settings for application projects.

Syntax

```
configapp
```

List name and description for available configuration parameters for the application projects. Following configuration parameters 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

```
configapp [OPTIONS] -app <app-name> <param-name>
```

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

```
configapp [OPTIONS] -app <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
-set	Set the configuration parameter value to new <value>.
-add	Append the new <value> to configuration parameter value.
-remove	Remove <value> from the configuration parameter value.
-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.

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

Example(s)

```
configapp
```

Return the list of all the configurable options for the application.

```
configapp -app test build-config
```

Return the current build configuration.

```
configapp -app test build-config release
```

Set the current build configuration to release.

```
configapp -app test define-compiler-symbols FSBL_DEBUG_INFO
```

Add the define symbol FSBL_DEBUG_INFO to be passed to the compiler.

```
configapp -app test -remove define-compiler-symbols FSBL_DEBUG_INFO
```

Remove the define symbol FSBL_DEBUG_INFO to be passed to the compiler.

```
configapp -app test compiler-misc {-pg}
```

Append the -pg flag to compiler misc flags.

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

Set flags specified to compiler misc.

```
configapp -app test -info compiler-optimization
```

Display more information about possible values/operation and default operation for compiler-optimization.

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

HSI Commands

XSCT provides higher level abstraction commands for Hardware Software Interface (HSI) commands and you normally do not have to run the HSI commands in XSCT. However, if there is a need, you can run HSI commands by prefixing `hsi` to each HSI command. For example, `hsi open_hw_design`.

***Note*:** You can use only one set of the commands at a time on any design. Interleaving both sets of commands will lead to internal errors. For example, a design opened with `hsi open_hw_design` cannot be closed using `closehw`.

- XSCT keeps track of the open designs (both software and hardware). The table below lists the HSI commands and their corresponding XCST commands.

HSI Command	Corresponding XSCT Command
`hsi open_hw_design`	`openhw`
`hsi close_hw_design`	`closehw`
`hsi open_sw_design`	`openbsp`
`hsi close_sw_design`	`closebsp`

- Any modifications done to the software design or the BSP, using HSI commands, are stored in memory. Before using any other XSCT commands, you should run the `updatemss` XSCT command, to ensure that these modifications are stored in the `.mss` file of the BSP.

For more details on the HSI commands and their usage, refer to the *Generating Basic Software Platforms Reference Guide* (**UG1138**).

XSCT Use Cases

As with Xilinx® Software Development Kit (Xilinx SDK) , the first step to use Xilinx Software Command-line Tool (XSCT) involves selecting a workspace. For creating and managing projects, XSCT launches Xilinx SDK in background. XSCT workspaces can be seamlessly used with Xilinx SDK and vice-versa.

***Note*:** At any given point of time, a workspace can either be used only from Xilinx SDK or XSCT.

The following is a list of use cases describing how you can use the tool to perform common tasks:

- Running Tcl Scripts
- Creating an Application Project Using an Application Template
- Modifying BSP Settings
- Changing Compiler Options of an Application Project
- Working with Libraries
- Creating a Bootable Image and Program the Flash
- Switching Between XSCT and Xilinx SDK Developement Environment
- Performing Standalone Application Debug
- Running an Application in Non-Interactive Mode
- Debugging a Program Already Running on the Target
- Using JTAG UART
- Debugging Applications on Zynq UltraScale+ MPSoC

Changing Compiler Options of an Application Project

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

```
setws /tmp/wrk/workspace
createhw -name hw0 -hwspec /tmp/wrk/system.hdf
createapp -name test -app {Empty Application} -proc psu_cortexa53_0 -
hwproject hw0 -os standalone
importsources -name test -path /tmp/wrk/testsources/test/
configapp -app test -add compiler-misc {-std=c99}
projects -build
```

Creating an Application Project Using an Application Template

Below is an example XSCT session that demonstrates creating a FSBL project for an Cortex® -A53 processor.

Note: Creating an application project will create a BSP project by adding the necessary libraries. `FSBL_DEBUG_DETAILED` symbol is added to FSBL for debug messages.

```
setws /tmp/wrk/workspace
createhw -name hw0 -hwspec /tmp/wrk/system.hdf
createapp -name fsbl1 -app {Zynq MP FSBL} -proc psu_cortexa53_0 -hwproject
hw0 -os standalone
configapp -app fsbl1 define-compiler-symbols FSBL_DEBUG_DETAILED
projects -build
```

Creating a Bootable Image and Program the Flash

Below is an example XSCT session that demonstrates creating two applications (FSBL and Hello World). Further, create a bootable image using the applications along with bitstream and program the image on to the flash.

Note: Assuming the board to be zc702. Hence `-flash_type qspi_single` is used as an option in `program_flash`.

```
setws /tmp/wrk/workspace
createhw -name hw0 -hwspec /tmp/wrk/system.hdf
createapp -name fsbl -app {Zynq FSBL} -proc ps7_cortexa9_0 -hwproject hw0 -
os standalone
createapp -name hello -app {Hello World} -proc ps7_cortexa9_0 -hwproject
hw0 -os standalone
projects -build
exec bootgen -arch zynq -image output.bif -w -o BOOT.bin
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

Xilinx® 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). Users will need to 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 debug info.

Below 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 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

# Once 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 Zynq® UltraScale+™ MPSoC boots up in JTAG bootmode. The flow described here can be applied to other bootmodes too, with minor changes.

When Zynq® UltraScale+™ MPSoC boots up JTAG bootmode, all the A53 and R5 cores are held in reset. Users must clear resets on each core, before debugging on these cores. 'rst' command in XSCT 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 A53 #0 is the current target, rst -cores clears resets on all the A53 cores in APU.

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

***Note*:** Similar steps can be used for debugging applications on R5 cores and also on A53 cores in 32 bit mode. However, the A53 cores must be put in 32 bit mode, before debugging the applications. This should be done after POR and before the 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 remove 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.

Modifying BSP Settings

Below is an example XSCT session that demonstrates building a Hello World application to target the MicroBlaze™ processor. The STDIN & STDOUT OS parameters are changed to use the `MDM_0`.

Note: Once the BSP settings are changed, it is necessary to update the mss & regenerate the BSP sources to reflect the BSP changes in the source file before compiling.

```
setws /tmp/wrk/workspace
createhw -name hw0 -hwspec /tmp/wrk/system_mb.hdf
createapp -name hello -app {Hello World} -proc microblaze_0 -hwproject hw0 -
os standalone
configbsp -bsp hello_bsp stdin mdm_0
configbsp -bsp hello_bsp stdout mdm_0
updatemss -mss hello_bsp/system.mss
regenbsp -bsp hello_bsp
projects -build
```

Performing Standalone Application Debug

Xilinx® System Command-line Tool (XSCT) 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.

Below is an example XSCT session that demonstrates standalone application debug on Zynq® -7000 AP SoC. 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.hdf
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`, users can download another elf onto `A9 #1` and debug it, using similar steps. Note that, it's not necessary to re-connect to the hw_server, initialize the PS or configure the FPGA in such cases. Users can just 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. Below is a example SVF script:

```
# 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 XSCT. You can use any standard SVF player to play the SVF file.

To play a SVF file in 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>
```

Running an Application in Non-Interactive Mode

Xilinx® 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.hdf
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 XSCT commands and run them in an interactive or non-interactive mode. In the interactive mode, you can source the script at XSCT prompt. For example:

```
xsct% source xsct_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:

```
$ xsct xsct_script.tcl [args]
```

The script below provides a usage example of XSCT. 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 SDK workspace
setws /tmp/workspace
# Create a HW project
createhw -name hw1 -hwspec /tmp/system.hdf
# Create a BSP project
createbsp -name bsp1 -hwproject hw1 -proc ps7_cortexa9_0 -os standalone
# Create application project
createapp -name hello -hwproject hw1 -bsp bsp1 -proc ps7_cortexa9_0 -os
standalone \
-lang C -app {Hello World}
# Build all projects
projects -build
# Connect to a remote 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 XSCT and Xilinx SDK Development Environment

Below is an example XSCT session that demonstrates creating two applications using XSCT and modifying the BSP settings. After the execution, launch the Xilinx® SDK development environment and select the workspace created using XSCT, to view the updates.

Note: The workspace created in XSCT can be used from Xilinx SDK. However, at a time, only one instance of the tool can use the workspace.

```
setws /tmp/wrk/workspace
createhw -name hw0 -hwspec /tmp/wrk/system.hdf
createbsp -name bsp0 -proc ps7_cortexa9_0 -hwproject hw0 -os standalone
createapp -name hello0 -app {Hello World} -proc ps7_cortexa9_0 -hwproject
hw0 -bsp bsp0 -os standalone
createapp -name fsbl0 -app {Zynq FSBL} -proc ps7_cortexa9_0 -hwproject hw0 -
bsp bsp0 -os standalone
projects -build
```

Using JTAG UART

Xilinx® System Debugger Command-line Interface (XSDB) supports virtual UART through Jtag, which is useful when the physical Uart doesn't exist or is non-functional. To use Jtag UART, the SW application should be modified to redirect STDIO to the Jtag UART. Xilinx SDK 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 Xilinx SDK 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 will be sent to STDIN and any output from STDOUT will be 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.

Below is an example XSCT session that demonstrates how to use a JTAG terminal for STDIO.

```
connect
source ps7_init.tcl
targets -set -filter {name =~"APU"}
loadhw system.hdf
stop
ps7_init
targets -set -nocase -filter {name =~ "ARM*#0"}
rst -processor
dow <app>.elf
jtagterminal
con
jtagterminal -stop #after you are done
```

Below is an example XSCT session that demonstrates how to use the XSCT console as STDOUT for JTAG UART.

```
connect
source ps7_init.tcl
targets -set -filter {name =~"APU"}
loadhw system.hdf
stop
ps7_init
targets -set -nocase -filter {name =~ "ARM*#0"}
rst -processor
dow <app>.elf
readjtaguart
con
readjtaguart -stop #after you are done
```

Below is an example XSCT session that demonstrates how to redirect the STDOUT from JTAG UART to a file.

```
connect
source ps7_init.tcl
targets -set -filter {name =~"APU"}
loadhw system.hdf
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

Below is an example XSCT session that demonstrates creating a normal BSP and add XILFFS & XILRSA libraries to the BSP. Create a FSBL application thereafter.

Note: Normal BSP do not contain any libraries.

```
setws /tmp/wrk/workspace
createhw -name hw0 -hwspec /tmp/wrk/system.hdf
createbsp -name bsp0 -proc ps7_cortexa9_0 -hwproject hw0 -os standalone
setlib -bsp bsp0 -lib xilffs
setlib -bsp bsp0 -lib xilrsa
updatemss -mss bsp0/system.mss
regenbsp -bsp bsp0
createapp -name fsbl0 -app {Zynq FSBL} -proc ps7_cortexa9_0 -bsp bsp0 -
hwproject hw0 -os standalone
projects -build
```

Changing the OS version.

```
setosversion -bsp bsp0 -ver 5.2
```

Assigning a driver to an IP.

```
setdriver -bsp bsp0 -ip ps7_uart_1 -driver generic
```

Removing a library (removes xilrsa library from BSP).

```
removelib -bsp hello_bsp -lib xilrsa
```

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx 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:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

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