Xilinx Software Command-Line Tool (XSCT)

Reference Guide

UG1208 (v2016.2) June 8, 2016

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
June 08, 2016	2016.2	Initial public release

Table of Contents

XILINX
ALL PROGRAMMABLE™

Chapter 1

Xilinx Software Command-Line Tool (XSCT)

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 Software Development Kit (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 you with the 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 AP 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 platform management unit (PMU).
- Hardware definition (HDF) files are used to transfer the information about the hardware system that includes a processor to the Xilinx SDK. 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 Software Command-Line Tool (XSCT), then you also need to meet the hardware and software requirements that are specific to that capability.

The following table lists the hardware and software requirements for XSCT:

Hardware Requirements	CPU Speed	2.2 GHz minimum or higher; Hyper-threading (HHT) or Multi-core 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)
Supported Operating Systems ***NOTE:*** 32-bit machine support is now only available through Lab Tools and Hardware Server standalone product installers.	Windows	• Windows 7 SP1 (64-bit) • Windows 8.1 (64-bit) • Windows 10 Pro (64-bit)
	Linux	• Red Hat Enterprise Linux: – 6.6-6.7 (64-bit) – 7.0-7.1 (64-bit) • CentOS: – 6.7 (64-bit) – 7.1 (64-bit) • SUSE Linux Enterprise: – 11.4 (64-bit) – 12.0 (64-bit) • Ubantu Linux 14.04.3 LTS (64-bit) ***NOTE:*** Additional library installation required.
Disk Space	Based on the components selected during installation.	

Chapter 2

Installing and Launching XSCT

The Xilinx Software Command-Line Tool (XSCT) can be installed both as a part of the Xilinx SDK installer and 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

Xilinx Software Command-line Tool (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've 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've 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 using the batch mode, investigate the hardware and software, debug and run the project, all from the command line.

XSCT commands are broadly classified into the following categories. 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 Acccess
- SDK Projects
- HSI Commands

TIP:

- *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

Use these commands to connect to or disconnect from hw_server and list or select active targets.

The following is a list of connections commands:

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

connect

Connect to hw_server.

NOTE: The `connect` command in XSDB doesn't connect to a debug target on the hardware. You must use the `targets` command to list the available targets and select a debug target before issuing commands to the debug target.

If no options are used with the `connect` command, XSDB launches hw_server at localhost:3121 and connects to it. The `connect` command returns the channel ID of the connection to the hw_server.

Synopsis

```
connect [Options]
```

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`
`-list`	List open connections
`-set <channel-id>`	Set active connection

Returns

Depends on the options specified:

- `<channel-id>` of the new connection or error if the connection fails, when `-port`, `-host`, `-url`, `-new` options are specified.
- List of open channels or nothing when there are no open channels, when the `-list` option is specified.
- Nothing when the `-set` option is specified.

Example

Connect to `hw_server` on host `localhost` and port `3121`.

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

Connect to `TCF agent` on host `localhost` and port `3121`.

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

disconnect

Disconnect from hw_server. As with the connect command, disconnect does not close a connection to the debug target, but will close the connection to hw_server.

Synopsis

```
disconnect [Options]
```

Options

Option	Description
`disconnect`	Disconnect from active channel.
`disconnect <channel-id>`	Disconnect from specified channel.

targets

List targets or switch between targets.

Synopsis

```
targets [Options]
targets <target id>
```

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.
`-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 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>}.

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

List all targets.

```
targets
```

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

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

Set target with id 2 as the current target.

```
targets 2
```

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

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

Set current target to target with name starting with "MicroBlaze" and which is on 1st Jtag Device

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

gdbremote connect

Connect to GDB remote server.

Synopsis

```
gdbremote connect [options]<server>
```

Options

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

Returns

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

gdbremote disconnect

Disconnect from GDB remote server.

Synopsis

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

Returns

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

Target Registers

These commands can be used to read and write to the target registers.

The following is a list of registers commands:

- rrd
- rwr

These commands can access general purpose registers, system registers like ARM co-processor registers.

TIP:

IOU registers in Zynq-7000 AP SoC devices can be accessed by running these commands on 'APU target'. rrd can also read register definitions, instead of their values.

rrd

Read register definitions or values for the active target.

Synopsis

```
rrd [Options] [reg]
```

Options

Option	Description
`-defs`	Read register definitions instead of values.
`-no-bits`	Don't 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 can't be read or if an invalid register is specified.

Example

Read top level registers or groups.

```
rrd
```

Read register `r0`.

```
rrd r0
```

Read register `r0` in group `usr`.

```
rrd usr r8
```

Read definitions for top level registers or groups.

```
rrd -defs
```

rwr

Write the <value> to active target register specified by <reg>.

Synopsis

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

Returns

Nothing, if successful.

Error string, if an invalid register is specified or the register can't be written.

Example

Write `0x0` to register `r8`

```
rwr r8 0x0
```

Write `0x0` to register `r8` in group `usr`.

```
rwr usr r8 0x0
```

Program Execution

These commands can be used for program execution, target state, and disassembly.

The following is a list of running commands:

- state
- stop
- con
- stp
- nxt
- stpi
- nxti
- stpout
- dis
- print
- locals
- backtrace
- profile

state

Display the current state of the target.

Synopsis

```
state
```

Returns

Return the current execution state of target.

stop

Suspend execution of active target.

Synopsis

```
stop
```

Returns

Nothing, if the target is suspended.

Error string, if the target is already stopped or can't be stopped. An info message is printed on the console when the target is suspended

con

Resume execution of active target.

Synopsis

```
con [Options]
```

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 can't be resumed or does not halt within timeout, after being resumed. An info message is printed on the console when the target is resumed.

Example

Resume execution of the active target from address `0x100000`.

```
con -addr 0x100000
```

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

```
con -block
```

Resume execution of the active target and wait until the target stops or until the 5 sec timeout is reached.

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

stp

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.

Synopsis

```
stp [count]
```

Options

Option	Description
`[count]`	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 can't be resumed. An info message is printed on the console when the target stops at the next address.

nxt

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.

Synopsis

```
nxt [count]
```

Options

Option	Description
`[count]`	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 can't be resumed. An info message is printed on the console when the target stops at the next address.

stpi

Execute a single machine instruction. If the instruction is a function call, stop at the first instruction of the function code.

Synopsis

```
stpi [count]
```

Options

Option	Description
`[count]`	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 can't be resumed. An info message is printed on the console when the target stops at the next address.

nxti

Step over a single machine instruction. If instruction is function call, execution continues until control returns from the function.

Synopsis

```
nxti [count]
```

Options

Option	Description
`[count]`	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 can't be resumed. An info message is printed on the console when the target stops at the next address.

stpout

Resume execution of current target until control returns from current function.

Synopsis

```
nxti [count]
```

Options

Option	Description
[count]	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 string, if the target is already running or can't be resumed. An info message is printed on the console when the target stops at the next address.

dis

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.

Synopsis

```
dis <address> <num>
```

Returns

Disassembled instructions if successful.

Error string, if the target instructions can't be read

Example

Disassemble an instruction at the current PC value.

```
dis
```

Disassemble two instructions at the current PC value.

```
dis pc 2
```

Disassemble two instructions at address 0x0.

```
dis 0x0 2
```

print

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`.

Synopsis

```
print [Options] [expression]
```

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.
`-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

- Expression value(s), if no option or `-add` is used.
- Expression definition(s), if `-defs` option is used.
- Nothing, if `-remove` or `-set` options are used.
- Error string, if expression value can't be read or set.

Example

Return the value of variable `Int_Glob`.

```
print Int_Glob
```

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

```
print -a Microseconds
```

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

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

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

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

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 tmp_var.var1.array1[0]
```

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

```
print
```

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

```
print -defs
```

Set the value of the variable `Int_Glob` to 23.

```
print -set Int_Glob 23
```

Remove the expression `Microseconds` from auto expression list.

```
print -remove Microseconds
```

Return the value of CPU register `r1`.

```
print {r1}
```

locals

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`.

Synopsis

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

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

- Variable value(s), if no option is used.
- Variable definition(s), if -defs option is used.
- Nothing, when variable value is set.
- Error string, if variable value can't be read or set.

Example

Return the value of the local variable `Int_Loc`.

```
locals Int_Loc
```

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

```
locals
```

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

```
locals -defs
```

Set the value of the local variable `Int_Loc` to 23.

```
locals Int_Loc 23
```

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

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

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.

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

backtrace

Return stact trace for current target. Target must be stopped. Use debug information for best results.

Synopsis

```
backtrace
```

Returns

Stack Trace, if successful.

Error string, if Stack Trace can't be read from the target

profile

Configure and run the GNU profiler. The profiling needs to enabled while building bsp and application to be profiled.

Synopsis

```
profile [options]
```

Options

Option	Description
`-freq`	Sampling frequency.
`-scratchaddr`	Scratch memory for storing the profiling related data. It needs to be assigned carefully, as it should not overlap with the program sections.
`-out`	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 the 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

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

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

Output the profile data in `testgmon.out`.

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

Target Memory

The Xilinx System Debugger CLI (XSDB) Memory commands can be used to Read and Write to the target memory space.

The following is a list of memory commands::

- mrd
- mwr
- memmap

Note: The 'force' option can be used to with these commands to over-write access protection.

mrd

Read memory address of the active target.

NOTE: Select an APU target to access ARM DAP and MEM-AP address space.

Synopsis

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

Read <num> words 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.
-size <access-size>	Access size can be one of the values below: • b = Bytes accesses • h = Half-word accesses • w = Word accesses Default access size is w. Address will be aligned to access-size before reading memory.
-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

Returns

- Memory addresses and data in requested format, if successful.
- Error string, if the target memory can't be read.

Example

Read a word at 0x0

```
mrd 0x0
```

Read 10 words at 0x0

```
mrd 0x0 10
```

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

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

Read 3 bytes at address `0x1`

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

Read 2 half-words at address `0x2`

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

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

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

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 0x100
```

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 APR 0x04
```

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 AP1 0x80090088
```

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

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

mwr

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

Synopsis

```
memmap [Options]
```

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 Default access size is w. Address will be aligned to access-size before writing to memory.

Option	Description
`-bin`	Read binary data from a file and write it to target address space.
`-file <file-name>`	File from which binary data is read before writing 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-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

Returns

- Nothing, if successful.
- Error string, if the target memory can't be written.

Example

Write `0x1234` to address `0x0`

```
mwr 0x0 0x1234
```

Write 4 values from the list of values to address `0x0`

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

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

write 3 bytes from the list at address `0x1`

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

write 2 half-words from the list at address `0x2`

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

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

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

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 0x100 0x80000042
```

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-dap 0x04 0xf8000120
```

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 1 0x80090088 0x03186003
```

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

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

osa

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

Synopsis

```
osa [Options]
```

Options

Option	Description
`-disable`	Disable OS awareness for a symbol file. If this option is not specified, OS awareness is enabled. ***NOTE:*** `-fast-exec` and `-fast-step` options are not valid with `-disable` option.
`-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.

Returns

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

Example

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

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

Disable OSA for <symbol-file>.

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

memmap

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

NOTE: Only the memory regions previously added through `memmap` command can be removed.

Synopsis

```
memmap <Options>
```

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. ***NOTE:*** Only the memory regions previously added through memmap command can be removed.
`-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.

Returns

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

Example

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

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

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

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

Target Download FPGA/BINARY

These commands can be used to download elf/binary files to the target or configure FPGA.

The following is a list of download commands:

- dow
- elfverify
- fpga

dow

Download elf and binary files to the target.

Synopsis

```
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 the debug symbol info for the code. `<addr>` is the relative address, to which all the program sections are relocated.

Returns

Nothing.

elfverify

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

Synopsis

```
elfverify [options]<file>
```

Options

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

Returns

Nothing.

fpga

Configure FPGA with a bitstream.

NOTE: FPGA device should be selected through the `targets` command before running the **fpga** command.

Synopsis

```
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.
`-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.

Returns

Depends on the options used.

- -file, -partial - Nothing, if fpga is configured, or an error if the configuration failed.
- Configuration value.

Target Reset

These commands can be used to reset the target.

The following is a list of reset commands:

rst

rst

Reset the active target. This command can be used to reset a processor or a group of processors or the entire system.

Synopsis

```
rst [options]
```

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

These commands can be used to add, remove, enable, disable or list Breakpoints/Watchpoints. XSDB supports Breakpoints/Watchpoints at address and source line.

The following is a list of breakpoint commands:

- bpadd
- bpremove
- bpenable
- bpdisable
- bplist
- bpstatus

bpadd

Set a breakpoint/watchpoint.

NOTE: 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.

Synopsis

```
bpadd <options>
```

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. Type can be one of the following values: • `auto` = Auto - Breakpoint type is chosen by hw_server. This is the default type • `hw` = Hardware Breakpoint • `sw` = Software Breakpoint
`-mode <breakpoint-mode>`	Specify the access mode that will trigger the breakpoint. The mode can be a bitwise OR one of the following values: • `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.
`-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 advances breakpoint properties.

Returns

Breakpoint Id.

Example

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

```
bpadd -addr 0x100000
```

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

```
bpadd -addr &main
```

Set a Hardware Breakpoint at `test.c:23`.

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

Set a Read_Write Watchpoint on variable `fooVar`.

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

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

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

bpremove

Remove Breakpoints/Watchpoints.

Synopsis

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

Options

Option	Description
`<id-list>`	Remove the Breakpoints/Watchpoints specified by `<id-list>`. `<id-list>` is a list of breakpoint IDs, which are returned by the bpadd command. Breakpoint IDs can also be obtained through the bplist command.
`-all`	Remove all breakpoints.

Returns

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

Example

Remove breakpoint `0`.

```
bpremove 0
```

Remove breakpoints `1` and `2`.

```
bpremove 1 2
```

Remove all breakpoints.

```
bpremove -all
```

bpenable

Enable Breakpoints/Watchpoints.

Synopsis

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

Options

Option	Description
`<id-list>`	Enable the Breakpoints/Watchpoints specified by `<id-list>`. `<id-list>` is a list of breakpoint IDs, which are returned by the `bpadd` command. Breakpoint IDs can also be obtained through the bplist command.
`-all`	Enable all breakpoints.

Returns

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

Example

Enable breakpoint `0`.

```
bpenable 0
```

Enable breakpoints `1` and `2`.

```
bpenable 1 2
```

Enable all breakpoints.

```
bpenable -all
```

bpdisable

Disable Breakpoints/Watchpoints.

Synopsis

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

Options

Option	Description
`<id-list>`	Disable the Breakpoints/Watchpoints specified by `<id-list>`. `<id-list>` is a list of breakpoint IDs, which are returned by the `bpadd` command. Breakpoint IDs can also be obtained through the bplist command.
`-all`	Disable all breakpoints.

Returns

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

Example

Enable breakpoint `0`.

```
bpenable 0
```

Enable breakpoints `1` and `2`.

```
bpenable 1 2
```

Enable all breakpoints.

```
bpenable -all
```

bplist

List all the Beakpoints/Watchpoints along with brief status for each Breakpoint and the target on which it is set.

Synopsis

```
bplist
```

Options

None.

Returns

List of breakpoints

bpstatus

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

Synopsis

```
breakpoint <id>
```

Returns

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

JTAG UART

These commands can be used to connect to JTAG UART (MDM or ARM DCC).

The following is a list of stream commands:

- jtagterminal
- readjtaguart

jtagterminal

Start/Stop Jtag based hyper-terminal.

NOTE: Select a MDM or ARM processor target before runnning this command.

Synopsis

```
jtagterminal [options]
```

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

Returns

Socket port number.

readjtaguart

Start or Stop reading from JTAG UART.

NOTE: While running a script in non-interactive mode, output from Jtag uart might not be written to the log, until `readjtaguart -stop` is used.

Synopsis

```
readjtaguart [options]
```

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.

Returns

Nothing, if successful. Error string, if data can't be read from the Jtag Uart.

Example

Start reading from the Jtag Uart and print the output on stdout.

```
readjtaguart
```

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

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

Stop reading from the JTAG UART.

```
readjtaguart -stop
```

Miscellaneous

The following is a list of miscellaneous commands:

- loadhw
- unloadhw
- mdm_drwr
- mb_drwr
- mdm_drrd
- mb_drrd
- configparams
- version
- xsdbserver start
- xsdbserver stop
- xsdbserver disconnect
- xsdbserver version

loadhw

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.

Synopsis

```
loadhw [options]
```

Options

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

Returns

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

Example

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

```
targets -filter {name =~ "APU"}
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.

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

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 doesn't clear memory map explicitly set by users.

Synopsis

```
unloadhw
```

Returns

Nothing

mdm_drwr

Write to MDM Debug register.

Synopsis

```
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

Write to MDM Break/Reset Control register.

```
mdm_drwr 8 0x40 8
```

mb_drwr

Write to MicroBlaze debug register

Synopsis

```
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`	Specify user bscan port number. Default is 2.

Returns

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

Example

Write to MB Control register.

```
mb_drwr 1 0x282 10
```

mdm_drrd

Read from MDM debug register.

Synopsis

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

Read a MDM debug register. `<cmd>` is 8-bit MDM command to access a Debug register. `<bitlen>` is the register width.

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

Read `XMDC ID` register.

```
mdm_drrd 0 32
```

mb_drrd

Read from MicroBlaze Debug Register

Synopsis

```
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.

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

Read MB Status register.

```
mb_drrd 3 28
```

configparams

List, get or set configuration parameters.

Synopsis

```
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 might 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 represents the global default. Not all options support context specific values.

Returns

Depends on the options specified.

- <none> - List of paramters 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

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

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

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

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

version

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

Synopsis

```
version [option]
```

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.

Synopsis

```
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 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 incoming connections.
-port <port>	Specified 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

Start XSDB server listener using dynamically allocated port.

```
xsdbserver start
```

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

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

xsdbserver stop

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

Synopsis

```
xsdbserver stop
```

Returns

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

xsdbserver disconnect

Disconnect current XSDB server connection.

Synopsis

```
xsdbserver disconnect
```

Returns

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

xsdbserver version

Return XSDB command server protocol version.

Synopsis

```
xsdbserver version
```

Returns

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

JTAG Access

These commands can be used to perform raw JTAG shifts, get/set device properties, lock/unlock JTAG cable, etc.

The following is a list of jtag commands:

jtag targets

List JTAG targets or switch between JTAG targets.

Synopsis

```
jtag targets
```

List available JTAG targets

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

Select <target id> as active JTAG target.

Options

Option	Description
-set	Set current target to entry single entry in list. This is useful in combination with -filter option. An error 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.
-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-properrties	Returns a Tcl list of dict's 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.

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

List all targets.

```
jtag targets
```

List targets with name "arm_dap".

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

Set target with id 2 as the current target

```
jtag targets 2
```

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

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

List Jtag cables.

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

jtag sequence

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

Synopsis

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

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

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

```
sequence drshift [options] bits [data]
```

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

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

`sequence delay usec`

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

`sequence get_pin <pin>`

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

`sequence set_pin <pin> <value>`

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

`sequence atomic enable`

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

`sequence run [options]`

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

Option	Description
-binary	Format return value(s) as binary. The first bit shifted out is the least significant bit in the first byte returned.
-bits	Format return value(s) as binary text string. The first bit shifted out is the first character in the string.
-hex	Format return value(s) as hexadecimal text string. The first bit shifted out is the least significant bit of the first byte of the in the string.

Option	Description
`-single`	Combine all return values as a single piece of data. Without this option the return value is a list with one entry for every shift with `-capture` and every `get_pin`.

`sequence clear`

Remove all commands from sequence.

`sequence delete`

Delete sequence.

Returns

Jtag sequence object.

Example

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

jtag device_properties

Get/set device properties.

Synopsis

`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

Return Tcl dict containing device properties for idcode `0x4ba00477`.

```
jtag device_properties 0x4ba00477
```

Set device properties for idcode `0x4ba00477`.

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

jtag lock

Lock JTAG scan chain containing current JTAG target.

Synopsis

```
jtag lock [timeout]
```

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 sequence run` 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 sequence run` commands needs to be done without interruption.

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

Returns

Nothing

jtag unlock

Unlock JTAG scan chain containing current JTAG target.

Synopsis

```
jtag unlock
```

Returns

Nothing

jtag claim

Claim JTAG device.

Synopsis

```
jtag claim mask
```

This command will attempt 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 allows 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: 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.

Synopsis

```
jtag disclaim mask
```

Clear claim mask for current JTAG device.

Returns

Nothing.

jtag frequency

Get/set JTAG frequency.

Synopsis

```
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

- Nothing, if Jtag frequency is successfully set.
- Current Jtag frequency, if no arguments are specified.
- Supported Jtag frequencies, if `-list` option is used.
- Error string, if invalid frequency is specified or frequency can't be set.

jtag servers

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.

Synopsis

```
jtag servers [options]
```

Options

Option	Description
-list	
-format	
-open <server>	
-close <server>	

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

List opened servers and number of associated ports.

```
jtag servers
```

Connect to XVC server on host localhost port 10200.

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

Close XVC server for host localhost port 10200.

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

SDK Projects

SDK commands support creation of hardware, BSP and application projects. You can also use the commands to manage repositories, set toolchain preferences, and configure, build and run applications.

The following is a list of SDK project commands:

- openhw
- closehw
- openbsp
- closebsp
- updatemss
- getaddrmap
- getperipherals
- configbsp
- repo
- setlib
- getlibs
- removelib

- setdriver
- getdrivers
- setosversion
- getos
- regenbsp
- projects
- setws
- getws
- createhw
- createbsp
- createapp
- importprojects
- importsources
- getprojects
- deleteprojects
- configapp
- toolchain

openhw

Open a hardware design exported from Vivado. HDF/XML file or the hardware project created using `createhw` command can be passed as argument. PS initialization files are also generated, if not present.

PS7: ps7_init.c, ps7_init.h, ps7_init.tcl

PSU: psu_init.c, psu_init.h, psu_init.tcl

Synopsis

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

Options

None

Returns

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

Example

Open the hardware project ZC702_hw_platform

```
openhw ZC702_hw_platform
```

Open the hardware project corresponding to the system.hdf

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

closehw

Close a hardware design that is opened using `openhw` command.

HDF/XML file or the hardware project created using `createhw` command can be passed as argument.

Synopsis

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

Options

None

Returns

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

Example

Close the hardware project ZC702_hw_platform

```
closehw ZC702_hw_platform
```

Close the hardware project corresponding to the system.hdf

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

openbsp

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

Synopsis

```
openbsp <bsp project |mss file>
```

Options

None

Returns

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

Example

Open the BSP project 'hello_bsp'

```
openbsp hello_bsp
```

Open the bsp project corresponding to the system.mss

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

closebsp

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

Synopsis

```
closebsp <bsp-project |mss-project>
```

Options

None

Returns

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

Example

Close the BSP project 'hello_bsp'

```
closebsp hello_bsp
```

Close the bsp project corresponding to the system.mss

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

updatemss

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

Synopsis

```
updatemss [options]
```

Options

Option	Description
`-hw <hw-proj | hdf/xml file>`	Hardware project or hardware specification file.
`-mss <mss file>`	MSS file to be updated.

Returns

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

Example

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

```
updatemss -hw ZC702_hw_platform -mss system.mss
```

getaddrmap

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

Synopsis

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

Options

None

Returns

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

Example

Return the address map of peripheral connected to `ps7_cortexa9_0`. `hw1` is the hardware project, which can be created using command `createhw`.

```
getaddrmap hw1 ps7_cortexa9_0
```

Return the address map of peripheral connected to `ps7_cortexa9_0`. `system.hdf` is the hardware specification file exported from Vivado.

```
getaddrmap system.hdf ps7_cortexa9_0
```

getperipherals

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.

Synopsis

```
getperipherals <hw proj |hdf/xml file> [processor-instance]
```

Options

None

Returns

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

Example

Return a list of peripherals in the hardware design.

```
getperipherals system.hdf
```

Return a list of peripherals connected to processor CortexA9#0 in the hardware design.

```
getperipherals system.hdf ps7_cortexa9_0
```

configbsp

Configure settings for BSP projects.

Synopsis

```
configbsp [options][<parameter> [<value> ]]
```

Options

Option	Description
`-hw <hw-proj | hdf/xml file>`	Hardware project or hardware specification file.
`-bsp <bsp_prj | mss file>`	BSP project or mss file.
`-proc | -os | -lib <lib-name>`	Return the configurable parameters of processor/os/library 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, 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 an error in case unsupported parameter is specified.

Example

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

```
configbsp -os -hw zc702_hw_platform -bsp system.mss
```

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

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

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

```
configbsp -lib xilisf -hw system.hdf -bsp system.mss
```

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

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

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

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

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

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

repo

Get/set the currently used software repositories path. This command can be used to scan the repositories. It can also be used to get the list of OS/libs/drivers/apps from repository.

Synopsis

```
repo [options]
```

Options

Option	Description
`-set <path-list>`	Set the repository path and load all the software cores available. Multiple repository paths can be specified in the form of list.
`-get`	Get the repository path.
`-scan`	Scan the repository path. Used to scan the repository, when some changes are done.
`-os | -libs | -drivers | -apps`	Return the list of all the OS/libs/drivers/apps from the repository.

Returns

Depends on the arguments 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

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

```
repo -set /tmp/wrk/repo
```

Return a list of OS from the repo.

```
repo -os
```

Return a list of libraries from the repo.

```
repo -libs
```

setlib

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.

Synopsis

```
setlib [options]
```

Options

Option	Description
`-hw <hw-proj | hdf/xml file>`	Hardware project or hardware specification file.
`-bsp <bsp_prj | 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, returns an error.

Example

Add the xilffs library with version 2.0 to the BSP.

```
setlib -hw zc702_hw_platform -bsp hello_bsp -lib xilffs -ver 2.0
```

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

```
setlib -hw zc702_hw_platform -bsp hello_bsp -lib xilrsa
```

getlibs

Returns the list of libraries and their versions from BSP in tabular format.

Synopsis

```
getlibs [options]
```

Options

Option	Description
`-hw <hw-proj | hdf/xml file>`	Hardware project or hardware specification file.
`-bsp <bsp_prj | mss file>`	BSP project or mss file.

Returns

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

Example

Return the list of all libraries in the BSP.

```
getlibs -hw zc702_hw_platform -bsp hello_bsp
```

removelib

Remove the library from BSP.

Synopsis

```
removelib [options]
```

Options

Option	Description
-hw <hw-proj | hdf/xml file>	Hardware project or hardware specification file.
-bsp <bsp_prj | 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, returns an error.

Example

Remove the xilffs library from the BSP.

```
removelib -hw zc702_hw_platform -bsp hello_bsp -lib xilffs
```

setdriver

Set driver for a IP in the BSP.

Synopsis

```
setdriver [options]
```

Options

Option	Description
-hw <hw-proj | hdf/xml file>	Hardware project or hardware specification file.
-bsp <bsp_prj | mss file>	BSP project or mss file.
-ip <ip-name>	IP name for which driver needs to be set.
-driver <driver-name>	Driver name which needs to be added to the BSP.
-ver <lib-ver>	Version of the driver.

Returns

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

Example

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

```
setdriver -hw zc702_hw_platform -bsp hello_bsp -ip ps7_uart -driver generic -ver 2.0
```

getdrivers

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

Synopsis

```
getdrivers [options]
```

Options

Option	Description
-hw <hw-proj | hdf/xml file>	Hardware project or hardware specification file.
-bsp <bsp_prj | mss file>	BSP project or mss file.

Returns

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

Example

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

```
getdrivers -hw zc702_hw_platform -bsp hello_bsp
```

setosversion

Set OS version in the BSP.

Synopsis

```
setosversion [options]
```

Options

Option	Description
-hw <hw-proj | hdf/xml file>	Hardware project or hardware specification file.
-bsp <bsp_prj | 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, returns an error.

Example

Set the OS version 5.4 in the BSP.

```
setosversion -hw zc702_hw_platform -bsp hello_bsp -ver 5.4
```

Set the latest OS version from repo in the BSP.

```
setosversion -hw zc702_hw_platform -bsp hello_bsp
```

getos

Return OS details from the BSP.

Synopsis

```
getos [options]
```

Options

Option	Description
-hw <hw-proj | hdf/xml file>	Hardware project or hardware specification file.
-bsp <bsp_prj | mss file>	BSP project or mss file.

Returns

If successful, this command returns OS details. Otherwise, returns an error.

Example

Return the OS details in the BSP.

```
getos -hw zc702_hw_platform -bsp hello_bsp
```

regenbsp

Regenerate the BSP sources.

Synopsis

```
regenbsp [options]
```

Options

Option	Description
-hw <hw-proj | hdf/xml file>	Hardware project or hardware specification file.
-bsp <bsp_prj | mss file>	BSP project or mss file.

Returns

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

Example

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

```
regenbsp -hw ZC702_hw_platform -bsp hello_bsp
```

projects

Build or clean a BSP/application project or all projects in the workspace.

Synopsis

```
projects [options]
```

Options

Option	Description
`-build | -clean`	Build or clean projects.
`-type <project-type>`	Project type can be: • `all` • `bsp` • `app` Default type is `all`.
`-name <project-name>`	Project name that should be built.

Returns

If successful, this command returns nothing. Otherwise, returns an error if invalid options are used or if the project can't be built.

Example

Build the BSP project "hello_bsp"

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

Build all the projects in the current workspace.

```
projects -build
```

Clean all the application projects in the current workspace.

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

Clean all the projects in the current workspace.

```
projects -clean
```

setws

Set Xilinx SDK workspace to `<path>`, for creating projects. If `<path>` doesn't exist, then the directory is created. If `<path>` is not specified, then the current directory is used.

Synopsis

```
setws [options] [path]
```

Options

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

Returns

If the workspace is set successfully, this command returns nothing. Returns an error if the path specified is a file.

Example

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

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

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

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

getws

Return the current Xilinx SDK workspace.

Synopsis

```
getws
```

Options

None

Returns

Current workspace.

Example

Return the current Xilinx SDK workspace.

```
getws
```

createhw

Create a hardware project.

Synopsis

```
createhw [options]
```

Options

Option	Description
`-name <project-name>`	Name of the project that needs to be created.
`-hwspec <hardware specification file>`	Path to the hardware specification file for creating a hardware project.

Returns

If the hardware project is created successfully, this command returns nothing. Returns an error if invalid options are used or if the project cannot be created.

Example

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

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

createbsp

Create a BSP project.

Synopsis

```
createbsp [options]
```

Options

Option	Description
`-name <project-name>`	Name of the project that needs to be created.
`-proc <processor-name>`	Processor instance that should be used for creating BSP project.
`-hwproject <hardware project name>`	HW 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 the mss file.
`-arch <32 | 64>`	Processor architecture 32 or 64-bit. This option is used to build the project with 32/64 bit tool chain. 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 can't be created.

Example

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
```

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 -proc ps7_cortexa9_0 -os standalone
```

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

```
createbsp -name bsp1 -hwproject hw1 -proc ps7_cortexa9_0 -mss system.mss
```

Create a BSP project with name `bsp1` for `psu_cortexa53_0` using a 32-bit toolchain.

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

createapp

Create an application project.

Synopsis

```
createapp [options]
```

Options

Option	Description
`-name <project-name>`	Name of the project that needs to be created.
`-app <template-application-name>`	Name of the template application. Default is "Hello World". This option is valid only for creating application projects.
`-proc <processor-name>`	Processor instance that should be used for creating application/BSP project. This option is valid only for creating application/bsp projects.
`-hwproject <hardware project name>`	HW project for which the application or bsp project should be created This option is valid only for creating application/BSP projects.
`-bsp <BSP project name>`	BSP project for which the application project should be created. This option is valid only for creating application projects. If this option is not specified, a default bsp is created.
`-os <OS name>`	OS type for the application project. Default type is standalone.
`-arch <32 | 64>`	Processor architecture 32 or 64-bit. This option is used to build the project with 32/64 bit tool chain. This is valid only for A53 processors, defaults to 32-bit for other processors.
`-lang <programming language>`	Programming language can be C or C++.

Returns

Nothing, if the application project is created successfully. Error string, if invalid options are used or if the project can't be created.

Example

Create a Hello World application project with name `hello1` for processor `ps7_cortexa9_0`.

```
createapp -name hello1 -app {Hello World} -bsp bsp1 -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 fsbl1 -app {Zynq FSBL} -hwproject hw1 -proc ps7_cortexa9_0
```

Create an empty C++ application project with name `e1`.

```
createapp -name e1 -app {Empty Application} -hwproject hw2 -proc microblaze_0 -lang c++
```

Create a Hello World application project with name `hello2` for processor `psu_cortexa53_0` with 32-bit toolchain.

```
createapp -name hello2 -app {Hello World} -hwproject hw1 -proc psu_cortexa53_0 -arch 32
```

importprojects

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

Synopsis

```
importprojects <path>
```

Options

None

Returns

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

Example

Import Xilinx SDK project(s) into the current workspace.

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

importsources

Import sources from a path to application project in workspace.

Synopsis

```
importsources [options]
```

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 can't be imported.

Example

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

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

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

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

getprojects

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

Synopsis

```
getprojects [options]
```

Options

Option	Description
`-type <project-type>`	Project type can be: • `all` • `bsp` • `app` Default type is `all`.

Returns

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

Example

Return the list of hardware projects.

```
getprojects -type hw
```

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

```
getprojects
```

deleteprojects

Delete project(s) from the workspace.

Synopsis

```
deleteprojects [options]
```

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 can't be deleted.

Example

Delete the `hello1` project from the disk.

```
deleteprojects -name hello1
```

Delete the `hello1` project from workspace only.

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

configapp

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

Synopsis

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

Options

Option	Description
-app <app-name>	Application name for which the parameter should be configured.
<param-name>	Configuration parameter name.
-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><parameter value> - Nothing if the value is set, or an error in case unsupported parameter is specified.

Example

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

```
xsct% configapp
  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 the program sections
  undef-compiler-symbols        Undefine symbols. Ex. MYSYMBOL
```

Set the current build configuration to release.

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

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

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

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

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

Append the –pg flag to compiler misc flags.

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

Set flags specified to compiler misc.

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

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

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

toolchain

Return a list of available toolchains and supported processor types.

Synopsis

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

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.

Example

Return the list of all supported toolchains.

```
xsct% toolchain
  Name            Supported CPU Types
  ====            ===================
  Code Sourcery   ps7_cortexa9
  Linaro          ps7_cortexa9, psu_cortexa53, psu_cortexa53_x32, psu_cortexr5
  Xilinx          microblaze, psu_pmu
```

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**).

Working with XSCT

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

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.

Sample script

```
# 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
```

Creating an Application Project Using an Application Template

Create 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.

Commands

```
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
```

Modifying BSP Settings

Build 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.

Commands

```
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 -hw hw0 -bsp hello_bsp stdin mdm_0
configbsp -hw hw0 -bsp hello_bsp stdout mdm_0
updatemss -hw hw0 -mss hello_bsp/system.mss
regenbsp -hw hw0 -bsp hello_bsp
projects -build
```

Changing Compiler Options of an Application Project

Create an empty application for Cortex A53 processor, by adding the compiler option `-std=c99`.

Commands

```
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
```

Adding Libraries to BSP

Create a normal BSP and add XILFFS & XILRSA libraries to the BSP. Create a FSBL application thereafter.

NOTE: Normal BSP do not contain any libraries.

Commands

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

Similar Usage

Changing the OS version.

```
setosversion -hw hw0 -bsp bsp0 -ver 5.2
```

Assigning a driver to an IP.

```
setdriver -hw hw0 -bsp bsp0 -ip ps7_uart_1 -driver generic
```

Removing a library (removes xilrsa library from BSP).

```
setdriver -hw hw0 -bsp bsp0 -ip ps7_uart_1 -driver generic
```

Creating a Bootable Image and Program the Flash

Create 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`.

Commands

```
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
```

Switching Between XSCT and Xilinx SDK Development Environment

Create two application using XSCT and modify the BSP settings. Launch the 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.

Commands

```
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
```

Performing Standalone Application Debug

Xilinx System Debugger Command-line Interface (XSDB) can be used to debug standalone applications on one or more processor cores simultaneously. The first step involved in debugging is to connect to hw_server and select a debug target. You can now reset the system/processor core, initialize the PS if needed, program the FPGA, download an elf, set breakpoints, run the program, examine the stack trace, view local/global variables.

Help for each of the xsdb commands can be viewed by running "help <command>" or "<command> -help" in XSDB console. All the available XSDB commands can be listed by running "help commands".

Below is an example XSDB 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

xsdb% 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

xsdb% targets
  1  APU
     2  ARM Cortex-A9 MPCore #0 (Running)
     3  ARM Cortex-A9 MPCore #1 (Running)
  4  xc7z020
xsdb% targets 2
# Reset the system before initializing the PS and configuring the FPGA

xsdb% 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

xsdb% 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
xsdb% loadhw ZC702_HwPlatform/system.hdf
design_1_wrapper

# Source the ps7_init.tcl script and run ps7_init and ps7_post_config commands
xsdb% source ZC702_HwPlatform/ps7_init.tcl
xsdb% ps7_init
xsdb% ps7_post_config

# Download the application program
xsdb% 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()
xsdb% bpadd -addr &main
0

# Resume the processor core
xsdb% con

# Info message is displayed when the core hits the breakpoint
```

```
xsdb% Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0x1005a4 (Breakpoint)

# Registers can be viewed when the core is stopped
xsdb% 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
xsdb% mrd 0xe000d000
E000D000:   800A0000

# 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: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0x1005b0 (Step)

# View stack trace
xsdb% 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
xsdb% bpadd -addr &exit
1
xsdb% con
Info: ARM Cortex-A9 MPCore #0 (target 2) Running
xsdb% Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0x103094 (Breakpoint)
xsdb% 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.

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
```

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
```

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.

Using 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
```

Using XSDB 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
```

Redirecting 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
```

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 XSDB can be used to clear the resets. 'rst -processor' clears reset on an individual processor core. 'rst -cores' clears resets on all the processor cores in the group (ex. 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 XSDB 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
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
```

XILINX
ALL PROGRAMMABLE™

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Please Read: Important Legal Notices