



Control File User Guide

Imperas Software Limited

Imperas Buildings, North Weston,
Thame, Oxfordshire, OX9 2HA, UK
docs@imperas.com



Author:	Imperas Software Limited
Version:	0.15
Filename:	OVP_Control_File_User_Guide.doc
Project:	Control File User Guide
Last Saved:	Wednesday, 22 April 2015

Copyright Notice

Copyright © 2015 Imperas Software Limited All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

IMPERAS SOFTWARE LIMITED., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents


1	Preface.....	5
1.1	Notation.....	5
1.2	Related Documents	5
2	Introduction.....	6
3	Use of Simulation Control Files	7
3.1	Control File Format.....	7
3.2	Summary of keywords	8
3.3	Specifying control files on the Imperas simulator command line	9
3.4	Specifying control files with the IMPERAS_TOOLS environment variable.....	9
3.5	Keyword descriptions	10
3.5.1	-batch <TCL command file>	10
3.5.2	-callcommand <cmdstring>	10
3.5.3	-cpuflags [proc=]<integer>	10
3.5.4	-debugprocessor <target>.....	11
3.5.5	-elfusevma [<proc>]	11
3.5.6	-enabletools <target>	11
3.5.7	-enabletoolspse <target>	12
3.5.8	-excludem <string>	12
3.5.9	-extlib <target>=<library>	12
3.5.10	-finishafter <integer>	12
3.5.11	-finishtime <decimal>.....	13
3.5.12	-gdbcommandfile [proc=]<path>.....	13
3.5.13	-gdbconsole	13
3.5.14	-gdbpath [proc=]<path>	13
3.5.15	-idebug.....	13
3.5.16	-interactive.....	13
3.5.17	-debugpseconstructors	14
3.5.18	-loadphysical [proc].....	14
3.5.19	-modeldiags [model=]value.....	14
3.5.20	-modelrecorddir <path>	14
3.5.21	-modelreplaydir <path>	15
3.5.22	-mpdconsole	15
3.5.23	-nowait	15
3.5.24	-nowarnings.....	15
3.5.25	-objectloader <path>	15
3.5.26	-o0.....	15
3.5.27	-program or -objfile or -objfileuseentry [proc=]<path>	15
3.5.28	-objfilenoentry [proc=]<path>	16
3.5.29	-override target=value	16
3.5.30	-port <integer>	16
3.5.31	-quantum	16
3.5.32	-quantumseed <integer>	17

3.5.33	-quiet	17
3.5.34	-reset or -startaddress [proc=]address	17
3.5.35	-searchpath	17
3.5.36	-showbuses	17
3.5.37	-showcommands	17
3.5.38	-showdomains	17
3.5.39	-showoverrides	17
3.5.40	-substitutepath <path>=<path>	17
3.5.41	-symbolfile [proc=]<path>	18
3.5.42	-timeprecision <integer>	18
3.5.43	-trace [proc]	18
3.5.44	-traceafter [proc=]<integer>	18
3.5.45	-tracebuffer	18
3.5.46	-tracechange	18
3.5.47	-tracereg	19
3.5.48	-traceshareddata	19
3.5.49	-traceshowicount	19
3.5.50	-variant [proc=]string	19
3.5.51	-verbose	19
3.5.52	-verbosedict	19
3.5.53	-vlnvmap v/l/n/v=v/l/n/v	19
3.5.54	-vlnvroot <path>	19
3.5.55	-wallclockfactor <decimal>	19
3.5.56	-wallclock	20
3.5.57	-werror	20

1 Preface

This document describes the capabilities of the Control file to modify configuration of a virtual platform executable and control the Simulator behavior.

1.1 Notation

 Code	Text representing a code extract or command line
<i>keyword</i>	A word with special meaning.

1.2 Related Documents

- OVPSim and CpuManager User Guide
- Imperas Installation Guide

2 Introduction

Imperas simulation technology enables very high performance simulation, debug and analysis of platforms containing multiple processors and peripheral models.

The *Control File* allows control of the simulation and models when the user has no access to the platform source code or does not wish to recompile it.

Most commands documented here are also available on the Imperas Simulator command line.

The Control File lets the user modify behavior in the following ways:

- Simulator behavior:
 - Finish times
 - Quantum size
 - Time precision
 - Verbosity and message control.
- Connecting to a remote debugger
- Setting or overriding model parameters
- Loading programs
- Loading semihost libraries, analysis and verification packages

Note that depending on the particular product that you downloaded, some control file features might be missing. However the `-help` option will correctly list the features present in your product.

3 Use of Simulation Control Files

3.1 Control File Format

A control file is written in plain text.

An entry begins with ‘-‘ (a second ‘-‘ is optional but has no significance) then a case-insensitive keyword optionally followed by space separated values or a single targeted value. Target and value are separated by ‘=’.

```
entry : -<keyword>  
      | -< keyword ><space><value>[<space><value>]  
      | -< keyword ><space><target>=<value>
```

A target is the hierarchical name of a platform, processor, peripheral model, plug-in or parameter. In some cases the target can be omitted and the value applied to all appropriate targets. Thus in a single processor platform, the target is optional for any entry that refers to the processor.

A value can be a path to a file, an integer, decimal or string. A string containing spaces must be enclosed in “quotes”.

```
value: <integer>  
      | <decimal>  
      | <string>  
      | “<string with spaces>”
```

A path to a file can be relative to the working directory or absolute, and must be legal on the host operating system.

A control file can contain blank lines and comment lines beginning with #.

3.2 Summary of keywords

Note that the 3rd column specifies if the keyword is available in OVPsim. All keywords are available in CpuManager. Keyword availability varies in other products. Please use the –help keyword if in doubt.

keyword	argument	ovp	description
–batch	<path>	n	The debugger will execute this TCL file
–callcommand	<cmdstring>	n	Call a command in a processor or plugin
–cpuflags	[proc=]<integer>	y	Set the value of the processor model-specific control flag.
–debugprocessor	proc	y	Specify a processor to connect to an RSP debug port
–debugpseconstructors		n	Start the debugger before PSE constructors have run
–elfusevma	[proc]	y	Use ELF VMA addresses rather than LMA
–excludem	string	y	Suppress simulator messages with this prefix
–extlib	proc=library	n	Add an extension library
–enabletools	[proc]	n	Turn on all analysis tools
–enabletoolspse	[pse]	n	Turn on all PSE analysis tools
–fetchvalidate	[proc]	y	(Model development mode) check instruction fetches
–finishafter	[proc=]<integer>	y	Finish simulation after this many processor instructions.
–finishtime	double	y	Finish the simulation after this time (in seconds).
–gdbcommandfile	[proc=]<path>	y	Supply a command file to a connected gdb.
–gdbconsole		y	Pop up a gdb in a console window
–gdbgui		n	Start the graphical single-processor debugger
–gdbpath	[proc=]<path>	y	Set the gdb path for this processor
–gui		n	Start graphical multi-processor debugger
–loadphysical	[proc]	y	Load using physical addresses from the program file.
–help		y	Print control file help, then exit
–idebug		n	Request debugger gdb command prompt
–interactive		n	Request debugger TCL command prompt
–mi		n	Start interactive debugger in MI mode (for use with GUI)
–modeldiags	[model=]<integer>	y	Diagnostics for peripheral models(can be overridden)
–modelrecorddir	<path>	n	All models (with replay mode) save replay files here
–modelreplaydir	<path>	n	All models (with replay mode) read replay files from here
–mpdconsole		n	Pop up the multiprocessor debugger in a console window
–nowait		n	Accept RSP connections but do not wait for a connection.
–nowarnings		y	Suppress warning messages
–objectloader	<path>	n	Provide a new object loader to be used by the simulator
–objfileuseentry	[proc=]<path>	y	Load object file onto CPU. Set PC to object entry point.
–objfilenoentry	[proc=]<path>	y	Load object file onto CPU. No not set PC to object entry point.
–output	<path>	y	Write the simulator log to this file.
–override	tgt=value	y	Override a value in platform definition. e.g. des/inst=123
–parallel		n	Enable parallel simulation of up to 8 CPUs
–parallelmax		n	Enable parallel simulation of any number of CPUs
–parallelopt	value	n	Set options for parallel simulation
–port	<integer>	y	Connect this port to GDB or MPD
–program	[proc=]<path>	y	Load this program (same as objfileuseentry)
–quantum	double	y	Scheduler time quantum (in seconds)
–quantumdifftime	double	n	Start of quantum in which to search for non-determinism
–quantumseed	<integer>	y	Order the processors pseudo-randomly, using this seed.
–quiet		y	Suppress ‘Info’ messages
–reset	[proc=]address	y	Specify a processor’s start address
–searchpath	[proc=]<path>	n	Tell the debugger where to find source file(if they have been moved since compilation)

-showbuses		y	Print platform bus connections
-showcommands		y	List commands that can be called with callcommand
-showdomains		y	List each memory domain
-showoverrides		y	List all possible platform overrides, then exit
-startaddress	[proc=<address>	y	Specify a processor's start address
-stoponcontrolc		y	Simulator will stop when ^C is received
-substitutepath	<path>=<path>	n	Add to list of path substitutes (debug information)
-symbolfile	[proc=<path>	n	Add to list of executables to read for symbolic info
-timeprecision	<integer>	y	Set the simulator time precision
-trace	[proc]	y	Trace instructions as they are executed
-traceafter	[proc=<integer>	y	Start tracing instructions after this many instructions
-tracebuffer	[proc]	y	Enable the trace buffer
-tracechange	[proc]	y	Print registers changed by the current instruction
-tracecount	[proc=<integer>	y	Trace this number of instructions
-tracereg	[proc]	y	Dump registers after each instruction
-traceshareddata		y	Trace the data shared by vmirtWriteListeners
-traceshowicount	[proc]	y	Show instruction count with each instruction
-variant	[proc]=variant	y	Specify a different variant
-verbose		y	Print more simulator information
-verboosedict		y	Show code dictionary statistics
-vlnvmap	v/l/n/v=v/n/l/v	n	Add to list of VLNV substitutes
-vlnvroot	path	n	Add to the VLNV search path
-wallclock		y	Limit maximum simulation speed to the wallclock time
-wallclockfactor	<decimal>	y	Limit maximum simulation speed to wallclock time * factor
-werror		y	Treat warnings as errors

Example:

```
# A simulation control file "control.ic"

-extlib          plat1/cpul/ext1=vapTools
-override        plat1/cpul/ext1/functiontrace on

-objfileuseentry plat1/cpu2=boot.elf      # boot code
-symbolfile       plat1/cpul=linux.elf    # symbols for linux
-callcommand      plat1/cpul/dumpTLB      # show initial contents of TLB

# end of file
```

3.3 Specifying control files on the Imperas simulator command line

A control file can be specified to the Imperas simulator using the command line argument `-controlfile`. This argument can be repeated to specify more control files. e.g.

```
shell> Imperas.exe --vlnvname MipsMalta \
        -controlfile control1.ic \
        -controlfile control2.ic
```

3.4 Specifying control files with the IMPERAS_TOOLS environment variable

Control files can be specified using the environment variable `IMPERAS_TOOLS`. Filenames are separated by `:` (Linux) or `;` (Windows) This method can be used with CpuManager when the simulator is a shared object that has no command line e.g.

```
shell> export IMPERAS_TOOLS="control.ic:control2.ic"
shell> mySystemC.exe      # simulator using CpuManager
```

3.5 Keyword descriptions

3.5.1 –batch <TCL command file> ...

If the product supports interactive debug, the debugger will be started in TCL model and will execute the given script or scripts. If `-interactive` is also specified a prompt will be printed and the simulator will wait for TCL commands after executing the given scripts.

Summary of debugger keywords:

keyword	interactive	language	note
-batch <file>	n	TCL	execute this tcl file first
-idebug	y	GDB	start debugger with gdb prompt
-interactive	y	TCL	start debugger with TCL prompt
-debugpseconstructors	y	GDB	stop before PSE constructors run

3.5.2 -callcommand <cmdstring>

Call a command in a model or plugin. Commands are specific to the model or plugin. <cmdstring> must start with the hierarchical name of a command in a processor or plugin, followed by the command's arguments, if any, separated by spaces. Commands with arguments must be enclosed in quotes. Use `-showcommands` to get a list of available commands.

Use `-callcommand "command -help"` to learn about a command's arguments.

e.g.

```
-showcommands
```

```
-callcommand "plat1/proc1/debug -help"
```

```
-callcommand "plat1/proc1/ostrace/trace -on"
```

3.5.3 -cpuflags [proc=<integer>

e.g.

```
-cpuflags plat1/proc2=0xA3
```

or

```
-cpuflags 0xFF
```

Some processor models have test modes and other non-standard functionality controlled by this value, which the model retrieves using the `vmirtProcessorFlags()` function. The value specified using this command in the control file will override a value specified in the platform. The interpretation of the flags value is processor model specific.

3.5.4 -debugprocessor <target>

When using a single-processor debugger, such as gdb, in a multi-processor platform, you must specify which processor to debug. e.g.

```
-debugprocessor plat1/proc2 # connect gdb to this processor
```

Most processor models can be connected to the Imperas remote Multi-Processor debugger. This allows simultaneous debug of more than one processor in the same platform. In this case you should specify the `-port` option; the `-debugprocessor` option is not required. Please refer to the Imperas_Debgugger_User_Guide.

Some Imperas simulators are able to connect simultaneously to several debuggers in which case this command can be repeated. e.g.

```
# (don't debug 1 and 2)
-debugprocessor plat1/proc3 # connect 1st gdb to this processor
-debugprocessor plat1/proc4 # connect 2nd gdb to this processor
# (don't debug 5 and 6)
```

3.5.5 -elfusevma [<proc>]

e.g.

```
-elfusevma plat1/proc1
```

or

```
-elfusevma # there is only one processor in this platform
```

By default the ELF reader loads to the Load Memory Addresses (LMA) specified in the ELF file rather than the Virtual Memory Addresses (VMA). The LMA can differ from the VMA in ELF files where data is to be copied by a program's startup code, usually from ROM to RAM.

This option may be used to override the default behavior and use the VMA instead of the LMA. This would be needed if for some reason the startup code that performs the copy is not included in the application.

Note that in release 20130630 and earlier the ELF loader would load to the VMA rather than the LMA. Later releases use the LMA by default and this option may be used to get the previous behavior.

3.5.6 -enabletools <target>

e.g.

```
-enabletools
```

```
-enabletools plat1/proc*
```

Load all analysis tools for all processors or a subset of all processors. Please refer to the Imperas VAP Tools User Guide.

3.5.7 -enabletoolspse <target>

e.g.

```
-enabletools
```

```
-enabletools plat1/psel
```

Load all analysis tools for all peripheral models or a subset of all models. Please refer to the Imperas VAP Tools User Guide.

3.5.8 -excludem <string>

e.g.

```
-excludem ICM_ # suppress all ICM messages
```

Suppress simulator messages with this prefix or part of prefix. Use this to improve the readability of simulator output when too many messages are appearing.

3.5.9 -extlib <target>=<library>

e.g.

```
# The installed library will be given a made-up name
-extlib plat1/proc2=imperas.com/intercept/vapTools/1.0
-extlib plat1/proc3=myLocalDir/model # .so or .dll is assumed
```

```
# The installed library gets its name from here
-extlib plat1/proc2/vap1=imperas.com/intercept/vapTools/1.0
-extlib plat1/proc3/myOne=myLocalDir/model # .so or .dll is assumed
```

Load an extension library or plug-in onto an application processor. The value can be a VLNV (See Imperas Simulator Guide) of a library in the VLNV tree, or a path to your own shared object.

If the target completely matches a processor name (1st two examples), an instance name is chosen by the simulator. If the target name has an extra field (2nd two examples) then this name is taken by the extension library instance.

⇒ The extension library instance name is significant if the library has parameters that may be overridden or commands that can be called.

3.5.10 -finishafter <integer>

e.g.

```
-finishafter plat1/proc1=12374 # finish after this many instructions
```

Stop the simulation when the specified processor has executed this many instructions. Note that the round-robin scheduling might cause other processors to execute more or fewer than this number.

e.g.

```
-finishafter 10000 # finish after this many instructions
```

This number applies to all processors.

3.5.11 **-finishtime <decimal>**

e.g.

```
# stop after 2 and three quarter seconds
-finishtime 2.75
```

Finish the simulation after a maximum time in seconds. There is no default. Use this to limit the execution time of a simulation should it go wrong.

3.5.12 **-gdbcommandfile [proc=]<path>**

e.g.

```
-gdbconsole
-debugprocessor plat1/proc1
-gdbcommandfile plat1/proc1=mystartup.cmd
```

When using `-gdbconsole` specify this command file for execution by the gdb attached to the specified processor. Use this to set complex breakpoints or conditions prior to getting control of the debugger.

3.5.13 **-gdbconsole**

The simulator will pop-up a console window (or xterm) running a gdb connected to a specified processor (see `-debugprocessor`). The path to an appropriate gdb is usually specified in the processor model. See `-gdbpath` to change the gdb.

3.5.14 **-gdbpath [proc=]<path>**

e.g.

```
-gdbconsole
-debugprocessor plat1/proc4
-gdbpath plat1/proc4=/home/me/mine/gdb
```

or

```
-gdbpath /home/me/mine/gdb # apply to all processors
```

Override the path to the gdb used to debug this processor. Use this when the gdb specified in the processor model is incorrect, missing from your installation or when you are debugging a new processor model.

3.5.15 **-idebug**

If the product supports interactive debug and the platform code is not using STDIN on the simulator process, the simulator will print the debugger prompt on STDOUT and wait for debugger commands. At this point, code in peripheral models or application processors will have run. Use this to debug application code.

3.5.16 **-interactive**

If the product supports interactive debug and the platform code is not using STDIN on the simulator process, the simulator will print the debugger prompt on STDOUT and wait for TCL commands. At this point, code in peripheral models or application processors will have run. Use this to debug application code.

3.5.17 **–debugpseconstructors**

When used with `–batch`, `–interactive` or `–idebug`, this command prevents the peripheral model constructors from running before the debugger starts. Use this command when debugging peripheral model constructors. Note that until the peripheral model constructors have run, peripheral model user views will not be available.

3.5.18 **–loadphysical [proc]**

e.g.

```
–loadphysical plat1/proc1
```

or

```
–loadphysical    # there is only one processor in this platform
```

This directs the program loader to load code at the physical addresses in the program file instead of the logical addresses which is more usual. This is very unusual. Refer to the documentation of your compiler and linker for more information.

Note that in release 20130630 and earlier the ELF loader would load to the VMA rather than the LMA. This option could be used to work around this problem, but could cause additional issues with symbols. Later releases use the LMA by default and so this option is no longer needed in most cases.

3.5.19 **–modeldiags [model=]value**

e.g.

```
–modeldiags plat1/dmal=0xF    # turn on one model's diags
```

or

```
–modeldiags 0xFF                # all models, maximum output
```

Most peripheral models have diagnostic output which can be turned on using this keyword. Refer to the Imperas Peripheral Modelling Guide for mode information.

3.5.20 **–modelrecorddir <path>**

e.g.

```
–recorddir /tmp/model_records    # all models save their output in here
```

Some peripheral models are able to record all their external input for replay in subsequent simulations. Applied to all peripheral models with external inputs, this makes the simulation deterministic so that bugs or other events of interest happen at exactly the same simulated time. Without this feature, some bugs are impossible to reproduce. Note that all models with external (and hence asynchronous) inputs must use this feature to achieve determinism. Please refer to Record and Replay in the Peripheral modeling guide.

3.5.21 **-modelreplaydir <path>**

e.g.

```
-replaydir /tmp/model_records # all models get their input from here
```

Some peripheral models are able to record all their external input for replay in subsequent simulations. Applied to all peripheral models with external inputs, this makes the simulation deterministic so that bugs or other events of interest happen at exactly the same simulated time. Without this feature, some bugs are impossible to reproduce. Note that all models with external (and hence asynchronous) inputs must use this feature to achieve determinism. Please refer to Record and Replay in the Peripheral modeling guide.

3.5.22 **-mpdconsole**

The simulator will pop up a console (or xterm) containing the remote multiprocessor debugger, and wait for input from this console. This feature is useful for connecting the MPD to your SystemC platform or other third party simulator that uses OVP models.

3.5.23 **-nowait**

Used with `-port`, the `-nowait` option lets the simulator start executing instructions without connecting to a debugger, but continue listening for a connection during simulation.

3.5.24 **-nowarnings**

Suppress simulator output with the ‘Warning’ severity. Use this with care; it might hide important information about an error in your configuration.

3.5.25 **-objectloader <path>**

e.g.

```
-objectloader mydir/myLoader # .so or .dll extension is added automatically
```

Provide a loader to load your own object files. Please refer to the CpuManager User Guide for details of the object loader API.

3.5.26 **-o0**

Turn off aggressive JIT code optimization. Use only if so instructed by Imperas.

3.5.27 **-program or -objfile or -objfileuseentry [proc=]<path>**

e.g.

```
-objfileuseentry plat1/proc2=mydir/myprogram.elf # program for one processor
```

or

```
-program mydir/myprogram.elf # program for all processors
```

(The three keywords are synonymous) Load a program for execution by an application processor. The program must be in a format supported by the simulator and compiled for the application processor. The simulator will set the PC of the application processor to the program’s entry point before starting. Debug and global symbols will be read by the

simulator for use in trace output and analysis tools. Use this option when simulating an application program without an OS or boot program.

3.5.28 **-objfilenoentry [proc=]<path>**

This is the same as `-objfileuseentry` but does NOT set the PC before starting simulation. Use this option when simulating a processor starting from its reset vector, or to load additional object files.

3.5.29 **-override target=value**

e.g.

```
-override plat1/proc1/variant=VAR99      # a model parameter
-override plat1/proc2/mips=150            # a simulator parameter
-override plat1/proc3/extlib1/libLog=log.txt # an extension library parameter
```

Set the value of a model or simulator parameter. Use the `-showoverrides` keyword to see a list of possible parameters.

3.5.30 **-port <integer>**

e.g.

```
-port 3000
```

Specify the port number for the simulator to listen on for connection to the debugger.

If the number is zero, a port is chosen from the operating system's pool and its number printed in the simulator log. To automate connection to the simulator, set the environment variable `IMPERAS_PORT_FILE` to a path, start the simulator then wait this file is written. It contains the chosen port number which can then be used by the debugger.

If you wish to debug application code on a single processor in your platform, use `-port` to specify a port number. If your platform contains more than one processor use `-debugprocessor` to specify which processor to debug. Having started your simulator and your gdb, use the gdb command:

```
target remote <hostname>:<portnumber>
```

to connect to the port.

By default, the simulator will wait for connection before any instructions are executed.

⇒ Some port numbers are used by other services. If the simulator reports this, try another number.

3.5.31 **-quantum**

e.g.

```
-quantum 0.002 # 2mS quantum; twice the default.
```

Sets the duration (in seconds) of the time-slice or quantum executed by each processor in turn. The default is 0.001s.

This keyword works if the platform calls `icmSimulatePlatform()`. If the simulation uses its own scheduler and calls `icmSimulate()` then changing the quantum has no effect. This is the case in a SystemC platform.

3.5.32 **-quantumseed <integer>**

e.g.

```
-quantumseed 1234
```

Use a random schedule order for each processor instead of the default round-robin. The integer specifies a pseudo random seed. This applies if `icmSimulatePlatform()` is used. If the simulation uses your own scheduler and `icmSimulate()`, it has no effect.

3.5.33 **-quiet**

Suppress simulator and model 'Info' messages.

3.5.34 **-reset or -startaddress [proc=]address**

e.g.

```
-reset plat1/proc1=0xBF00000 # start at this address
```

Set the PC of the specified processor to this value before starting the simulation. This will override a start address found in the application program, and override the reset vector in the model.

3.5.35 **-searchpath**

Tell the debugger where to find source files that have been moved since the program was compiled. In a multicore platform each processor could have source in a different location, so paths can be prefixed with the processor instance name. e.g.

```
-searchpath procA=/home/user/applicationSource
```

or

```
-searchpath localsource/src
```

3.5.36 **-showbuses**

Print a list of buses and their connections. Use this to check your design is correctly constructed.

3.5.37 **-showcommands**

Print a list of model commands that may be called with `-callcommand`.

3.5.38 **-showdomains**

Print a list of memory domains. Use this to check any dynamically mapped components.

3.5.39 **-showoverrides**

Print a list of platform, simulator and model parameters that can be set with `-override`.

3.5.40 **-substitutepath <path>=<path>**

e.g.

```
-substitutepath /home/develop/main=/local/debug/main
```

Use this to inform the debugger that source files used to compile the application being debugged are now at a different disk location. In the example, the application was compiled in /home/develop, so the debug symbols in the application program file will contain those references and the debugger will look there when debugging the program. If, since compiling the code, the files have been moved to /local/debug, the debugger will fail to find them. Use this to tell the debugger where they are now.

3.5.41 **-symbolfile [proc=]<path>**

e.g.

```
-symbolfile plat1/proc2=mydir/myprogram.elf # program for one processor
```

Sometimes the application program is not loaded explicitly by the simulator. It might be read from a simulated disk, or uncompressed by a boot loader. Use this command to pass the symbols to the simulator for use by intercept libraries, disassembly and tracing. The symbol file must be in a format accepted by one of the installed object loaders.

3.5.42 **-timeprecision <integer>**

Change the simulator time precision. This is the number to which time calculations are rounded. The default value is 1e-9.

3.5.43 **-trace [proc]**

e.g.

```
-trace p # tracing of all processors
```

or

```
-trace plat1/proc2 # tracing of one processor
```

Turn on basic instruction tracing.

⇒ Tracing will slow the simulation by several orders of magnitude.

3.5.44 **-traceafter [proc=]<integer>**

Begin tracing after this many instructions. Can be applied to one or all processors.

e.g.

```
-traceafter plat1/proc2=10000000 # tracing of one processor
```

or

```
-traceafter 100000000
```

3.5.45 **-tracebuffer**

Turn on a circular trace buffer which continually records the last 256 instructions executed. The trace buffer may be displayed from MPD and will also be written at the end of simulation.

3.5.46 **-tracechange**

When tracing is enabled, after each trace line, report any new register values.

3.5.47 -traceregs

When tracing is enabled, after each trace line, report all register values.

3.5.48 -traceshareddata

Report activity of the vmirtWriteListeners API. See the Vmi Run-time Programmers Reference.

3.5.49 -traceshowicount

When tracing is enabled, display the instruction number since the start of simulation in each trace line.

3.5.50 -variant [proc=]string

e.g.

```
-variant plat1/proc2=VAR99 # set the variant in a multi-processor platform  
or  
-variant VAR99 # set the variant in a single-processor platform
```

The values allowed are processor model specific. If an invalid value is specified a list of valid values will be displayed.

3.5.51 -verbose

Produce more information while starting, running and finishing simulation.

3.5.52 -verbosedict

Report information on the code dictionary at the end of simulation.

3.5.53 -vlnvmap v//n/v=v//n/v

e.g.

```
-vlnvmap ovpworld.org/processor/or1k/1.0=develop/processor/or1k/1.0
```

Replace a platform component without recompiling the platform.

3.5.54 -vlnvroot <path>

e.g.

```
-vlnvroot /home/develop/library
```

Add an additional directory to search for models.

3.5.55 -wallclockfactor <decimal>

e.g.

```
-wallclockfactor 2.0 # limit maximum simulation speed to twice real-time
```

Limit maximum simulation speed to this many times that of the wall clock (real time). Prevents time from racing forwards when there is no simulation activity. Use this if your simulation connects to a terminal or other GUI into which you wish to type. This applies if icmSimulatePlatform() is used. If the simulation uses your own scheduler and uses icmSimulate(), it has no effect.

3.5.56 -wallclock

Equivalent to `-wallclockfactor 1.0`

3.5.57 -werror

Treat warnings as if they are errors. A message with 'Error' severity during platform construction will prevent simulation. This option causes 'Warning' severity messages to be treated the same way.

##