



POWER8 Functional Simulator

Command Reference

*Understanding and using commands in the
POWER8 Functional Simulator environment*

Advance

10 April 2015
Version 1.1



© Copyright International Business Machines Corporation 2014 2015

Printed in the United States of America April 2015

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the design, sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

You may use this documentation solely for developing technology products compatible with Power Architecture®. You may not modify or distribute this documentation. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN “AS IS” BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at ibm.com®.

Version 1.1
10 April 2015



Contents

List of Figures	13
List of Tables	15
Revision Log	17
About this Document	19
Intended Audience	19
Using this Manual	19
Command Page Organization	19
Conventions	20
Related Documents	20
Help and Support	21
1. Introduction	23
1.1 Understanding and Using Simulator Commands	23
1.2 Managing a Simulated Machine	25
1.3 Overview of Command Structure and Syntax	26
1.4 Using the Command Pages	26
1.5 Accessing Help for Commands	27
1.6 Top-Level Simulator Commands	27
1.7 alias	29
1.7.1 Tcl Syntax	29
1.7.2 Description	29
1.7.3 Arguments	29
1.7.4 Examples	29
1.8 define	31
1.9 display	32
1.9.1 Tcl Syntax	32
1.9.2 Description	32
1.9.3 Examples	32
1.10 help or helprecursive	34
1.10.1 Tcl Syntax	34
1.10.2 Description	34
1.10.3 Examples	34
1.11 quit	35
1.11.1 Tcl Syntax	35
1.11.2 Description	35
1.11.3 Examples	35
1.12 simdebug	36
1.12.1 Tcl Syntax	36
1.12.2 Description	36
1.12.3 Arguments	36
1.12.4 Examples	36



1.13 simstop	37
1.13.1 Tcl Syntax	37
1.13.2 Description	37
1.13.3 Examples	37
1.14 version	38
1.14.1 Tcl Syntax	38
1.14.2 Description	38
1.14.3 Examples	38
2. Defining, Configuring, and Instantiating a Machine	39
2.1 define config	40
2.1.1 Tcl Syntax	40
2.1.2 Description	40
2.1.3 Arguments	40
2.1.4 Examples	40
2.1.5 Related Commands	40
2.2 define cpu	41
2.2.1 Tcl Syntax	41
2.2.2 Description	41
2.2.3 Arguments	41
2.2.4 Examples	41
2.2.5 Related Commands	41
2.3 define dup	42
2.3.1 Tcl Syntax	42
2.3.2 Description	42
2.3.3 Arguments	42
2.3.4 Examples	42
2.3.5 Related Commands	42
2.4 define list	43
2.4.1 Tcl Syntax	43
2.4.2 Description	43
2.4.3 Examples	43
2.4.4 Related Commands	43
2.5 define machine	44
2.5.1 Tcl Syntax	44
2.5.2 Description	44
2.5.3 Arguments	44
2.5.4 Examples	44
2.5.5 Related Commands	44
2.6 {configuration_object} config	45
2.6.1 Tcl Syntax	45
2.6.2 Description	45
2.6.3 Arguments	45
2.6.4 Examples	45
2.6.5 Related Commands	45
2.7 {configuration_object} display	46
2.7.1 Tcl Syntax	46
2.7.2 Description	46
2.7.3 Arguments	46

2.7.4 Examples	46
2.7.5 Related Commands	47
2.8 {configuration_object} exit	48
2.8.1 Tcl Syntax	48
2.8.2 Description	48
2.8.3 Examples	48
2.8.4 Related Commands	48
2.9 {configuration_object} query	49
2.9.1 Tcl Syntax	49
2.9.2 Description	49
2.9.3 Arguments	49
2.9.4 Examples	49
2.9.5 Related Commands	49
2.10 {configuration_object} quit	50
2.10.1 Tcl Syntax	50
2.10.2 Description	50
2.10.3 Examples	50
2.10.4 Related Commands	50
3. Configuring and Modifying Machine Properties	51
3.1 {machine} bogus disk cleanup	52
3.1.1 Tcl Syntax	52
3.1.2 Description	52
3.1.3 Examples	52
3.1.4 Related Commands	52
3.2 {machine} bogus disk display	53
3.2.1 Tcl Syntax	53
3.2.2 Description	53
3.2.3 Examples	53
3.2.4 Related Commands	53
3.3 {machine} bogus disk init	54
3.3.1 Tcl Syntax	54
3.3.2 Description	54
3.3.3 Arguments	54
3.3.4 Examples	55
3.3.5 Related Commands	55
3.4 {machine} bogus disk stat	56
3.4.1 Tcl Syntax	56
3.4.2 Description	56
3.4.3 Examples	56
3.4.4 Related Commands	56
3.5 {machine} bogus disk sync	57
3.5.1 Tcl Syntax	57
3.5.2 Description	57
3.5.3 Arguments	57
3.5.4 Examples	57
3.5.5 Related Commands	57
3.6 {machine} bogus net cleanup	58
3.6.1 Tcl Syntax	58

3.6.2 Description	58
3.6.3 Examples	58
3.6.4 Related Commands	58
3.7 {machine} bogus net init	59
3.7.1 Tcl Syntax	59
3.7.2 Description	59
3.7.2.1 Extended Description of Bogus Network Support	59
3.7.2.2 Setting up TUN/TAP on the Host System	59
3.7.2.3 Configuring systemsim-p8 Support for the Bogus Network	59
3.7.3 Arguments	60
3.7.4 Examples	60
3.7.5 Related Commands	60
3.8 {machine} bogushalt	61
3.8.1 Tcl Syntax	61
3.8.2 Description	61
3.8.3 Arguments	61
3.8.4 Examples	61
3.8.5 Related Commands	61
3.9 {machine} capi	62
3.9.1 Tcl Syntax	62
3.9.2 Description	62
3.9.3 Arguments	62
3.9.4 Examples	63
3.10 {machine} config_on	64
3.10.1 Tcl Syntax	64
3.10.2 Description	64
3.10.3 Examples	64
3.10.4 Related Commands	64
3.11 {machine} console create	65
3.11.1 Tcl Syntax	65
3.11.2 Description	65
3.11.3 Arguments	65
3.11.4 Examples	66
3.11.5 Related Commands	66
3.12 {machine} console destroy	67
3.12.1 Tcl Syntax	67
3.12.2 Description	67
3.12.3 Arguments	67
3.12.4 Examples	67
3.12.5 Related Commands	67
3.13 {machine} console disable	68
3.13.1 Tcl Syntax	68
3.13.2 Description	68
3.13.3 Arguments	68
3.13.4 Examples	68
3.13.5 Related Commands	68
3.14 {machine} console display buffered	69
3.14.1 Tcl Syntax	69
3.14.2 Description	69
3.14.3 Examples	69

3.14.4 Related Commands	69
3.15 {machine} console enable	70
3.15.1 Tcl Syntax	70
3.15.2 Description	70
3.15.3 Arguments	70
3.15.4 Examples	70
3.15.5 Related Commands	70
3.16 {machine} console list	71
3.16.1 Tcl Syntax	71
3.16.2 Description	71
3.16.3 Examples	71
3.16.4 Related Commands	71
3.17 {machine} console set display buffered	72
3.17.1 Tcl Syntax	72
3.17.2 Description	72
3.17.3 Arguments	72
3.17.4 Examples	72
3.17.5 Related Commands	72
3.18 {machine} cpu	73
3.18.1 Tcl Syntax	73
3.18.2 Description	73
3.18.3 Arguments	73
3.18.4 Examples	73
3.18.5 Related Commands	73
3.19 {machine} cycle	74
3.19.1 Tcl Syntax	74
3.19.2 Description	74
3.19.3 Arguments	74
3.19.4 Examples	74
3.19.5 Related Commands	74
3.20 {machine} display cycles	75
3.20.1 Tcl Syntax	75
3.20.2 Description	75
3.20.3 Examples	75
3.20.4 Related Commands	75
3.21 {machine} display features	76
3.21.1 Tcl Syntax	76
3.21.2 Description	76
3.21.3 Examples	76
3.22 {machine} display fpr, fpr_as_fp, fprs	77
3.22.1 Tcl Syntax	77
3.22.2 Description	77
3.22.3 Arguments	77
3.22.4 Examples	77
3.23 {machine} display gpr, gprs	79
3.23.1 Tcl Syntax	79
3.23.2 Description	79
3.23.3 Arguments	79
3.23.4 Examples	79

3.24 {machine} display instruction_count	81
3.24.1 Tcl Syntax	81
3.24.2 Description	81
3.24.3 Examples	81
3.24.4 Related Commands	81
3.25 {machine} display memory_size	82
3.25.1 Tcl Syntax	82
3.25.2 Description	82
3.25.3 Examples	82
3.25.4 Related Commands	82
3.26 {machine} display memorymap	83
3.26.1 Tcl Syntax	83
3.26.2 Description	83
3.26.3 Arguments	83
3.26.4 Examples	83
3.26.5 Related Commands	83
3.27 {machine} display nfpr, ngpr, mode, name	84
3.27.1 Tcl Syntax	84
3.27.2 Description	84
3.27.3 Examples	84
3.28 {machine} display number_of_MCMS	85
3.28.1 Tcl Syntax	85
3.28.2 Description	85
3.28.3 Examples	85
3.29 {machine} display slb, spr, tm, vmx, vmxr, vsxr	86
3.29.1 Tcl Syntax	86
3.29.2 Description	86
3.29.3 Arguments	86
3.29.4 Examples	86
3.30 {machine} dtranslate	88
3.30.1 Tcl Syntax	88
3.30.2 Description	88
3.30.3 Arguments	88
3.30.4 Examples	88
3.30.5 Related Commands	88
3.31 {machine} exit	89
3.31.1 Tcl Syntax	89
3.31.2 Description	89
3.31.3 Examples	89
3.31.4 Related Commands	89
3.32 {machine} go	90
3.32.1 Tcl Syntax	90
3.32.2 Description	90
3.32.3 Examples	90
3.33 {machine} interrupt	91
3.33.1 Tcl Syntax	91
3.33.2 Description	91
3.33.3 Arguments	91
3.33.4 Examples	91

3.34 {machine} itranslate	92
3.34.1 Tcl Syntax	92
3.34.2 Description	92
3.34.3 Arguments	92
3.34.4 Examples	92
3.34.5 Related Commands	92
3.35 {machine} load elf	93
3.35.1 Tcl Syntax	93
3.35.2 Description	93
3.35.3 Arguments	93
3.35.4 Examples	93
3.35.5 Related Commands	93
3.36 {machine} load linux	94
3.36.1 Tcl Syntax	94
3.36.2 Description	94
3.36.3 Arguments	94
3.36.4 Examples	94
3.36.5 Related Commands	94
3.37 {machine} load vmlinux	95
3.37.1 Tcl Syntax	95
3.37.2 Description	95
3.37.3 Arguments	95
3.37.4 Examples	95
3.37.5 Related Commands	95
3.38 {machine} load xcoff	96
3.38.1 Tcl Syntax	96
3.38.2 Description	96
3.38.3 Arguments	96
3.38.4 Examples	96
3.38.5 Related Commands	96
3.39 {machine} mcm	97
3.39.1 Tcl Syntax	97
3.39.2 Description	97
3.39.3 Arguments	97
3.39.4 Examples	97
3.39.5 Related Commands	97
3.40 {machine} memory display	98
3.40.1 Tcl Syntax	98
3.40.2 Description	98
3.40.3 Arguments	98
3.40.4 Examples	98
3.40.5 Related Commands	99
3.41 {machine} memory fread, freadcmp, freadgz	100
3.41.1 Tcl Syntax	100
3.41.2 Description	100
3.41.3 Arguments	100
3.41.4 Examples	100
3.41.5 Related Commands	100
3.42 {machine} memory fwrite, fwritecmp, fwritegz	101
3.42.1 Tcl Syntax	101



3.42.2 Description	101
3.42.3 Arguments	101
3.42.4 Examples	101
3.42.5 Related Commands	101
3.43 {machine} memory set	102
3.43.1 Tcl Syntax	102
3.43.2 Description	102
3.43.3 Arguments	102
3.43.4 Examples	102
3.43.5 Related Commands	102
3.44 {machine} mode	103
3.44.1 Tcl Syntax	103
3.44.2 Description	103
3.44.3 Arguments	103
3.44.4 Examples	103
3.45 {machine} process	104
3.45.1 Tcl Syntax	104
3.45.2 Description	104
3.45.3 Examples	104
3.46 {machine} quit	105
3.46.1 Tcl Syntax	105
3.46.2 Description	105
3.46.3 Examples	105
3.46.4 Related Commands	105
3.47 {machine} setargs	106
3.47.1 Tcl Syntax	106
3.47.2 Description	106
3.47.3 Arguments	106
3.47.4 Examples	106
3.47.5 Related Commands	106
3.48 {machine} stall	107
3.48.1 Tcl Syntax	107
3.48.2 Description	107
3.48.3 Examples	107
3.48.4 Related Commands	107
3.49 {machine} start_thread	108
3.49.1 Tcl Syntax	108
3.49.2 Description	108
3.49.3 Arguments	108
3.49.4 Related Commands	108
3.50 {machine} step	109
3.50.1 Tcl Syntax	109
3.50.2 Description	109
3.50.3 Arguments	109
3.50.4 Examples	109
3.50.5 Related Commands	109
3.51 {machine} stop_thread	110
3.51.1 Tcl Syntax	110
3.51.2 Description	110
3.51.3 Related Commands	110

3.52 {machine} thread	111
3.52.1 Tcl Syntax	111
3.52.2 Description	111
3.52.3 Arguments	111
3.52.4 Examples	111
3.52.5 Related Commands	111
3.53 {machine} tick	112
3.53.1 Tcl Syntax	112
3.53.2 Description	112
3.53.3 Arguments	112
3.53.4 Examples	112
3.53.5 Related Commands	112
3.54 {machine} to_cycle	113
3.54.1 Tcl Syntax	113
3.54.2 Description	113
3.54.3 Arguments	113
3.54.4 Examples	113
3.54.5 Related Commands	113
3.55 {machine} util dtranslate	114
3.55.1 Tcl Syntax	114
3.55.2 Description	114
3.55.3 Arguments	114
3.55.4 Examples	114
3.55.5 Related Commands	114
3.56 {machine} util dtranslate_wimg	115
3.56.1 Tcl Syntax	115
3.56.2 Description	115
3.56.3 Arguments	115
3.56.4 Examples	115
3.56.5 Related Commands	115
3.57 {machine} util itranslate	116
3.57.1 Tcl Syntax	116
3.57.2 Description	116
3.57.3 Arguments	116
3.57.4 Examples	116
3.57.5 Related Commands	116
3.58 {machine} util itranslate_wimg	117
3.58.1 Tcl Syntax	117
3.58.2 Description	117
3.58.3 Arguments	117
3.58.4 Examples	117
3.58.5 Related Commands	117
3.59 {machine} util ppc_disasm	118
3.59.1 Tcl Syntax	118
3.59.2 Description	118
3.59.3 Arguments	118
3.59.4 Examples	118
3.60 {machine} util stuff	119
3.60.1 Tcl Syntax	119
3.60.2 Description	119



POWER8 Functional Simulator

Advance

3.60.3 Arguments	119
3.60.4 Examples	119
3.60.5 Related Commands	119

List of Figures

Figure 1-1. Categories of Simulator Commands	24
Figure 1-2. Linux Boot of the POWER8 Functional Simulator with Console Display	25
Figure 1-3. Sample Command Page Content and Format	27
Figure 3-1. Calculating Ticks in the System	112





List of Tables

Table 1-1. POWER8 Functional Simulator Top-Level Commands 27



Revision Log

Each release of this document supersedes all previously released versions. The revision log lists all significant changes made to the document since its initial release. In the rest of the document, change bars in the margin indicate that the adjacent text was modified from the previous release of this document.

Revision Date	Pages	Description
10 April 2015		Version 1.1.
	27	Updated <i>Table 1-1 POWER8 Functional Simulator Top-Level Commands</i> .
	72	Added <i>Section 3.17 {machine} console set display buffered</i> .
	73	Corrected the example in <i>Section 3.18 {machine} cpu</i> .
	77	Added <i>Section 3.22 {machine} display fpr, fpr_as_fp, fprs</i> .
	82	Updated <i>Section 3.25 {machine} display memory_size</i> .
	84	Added <i>Section 3.27 {machine} display nfpr, ngpr, mode, name</i> .
	86	Added <i>Section 3.29 {machine} display slb, spr, tm, vmx, vmxr, vsxr</i> .
	88	Updated <i>Section 3.30 {machine} dtranslate</i> .
	91	Updated <i>Section 3.33 {machine} interrupt</i> .
	92	Updated <i>Section 3.34 {machine} itranslate</i> .
	97	Corrected the example in <i>Section 3.39 {machine} mcm</i> .
	98	Corrected the example in <i>Section 3.40 {machine} memory display</i> .
101	Added the append argument and corrected the example in <i>Section 3.42 {machine} memory fwrite, fwritecmp, fwritegz</i> .	
24 October 2014	—	Version 1.0. Initial release.



About this Document

The IBM® POWER8™ Functional Simulator was developed and refined in conjunction with several design projects built upon the IBM Power Architecture®. The POWER8 Functional Simulator enables hardware and software developers to simulate a POWER8 processor-based system to develop and enhance application support for this platform. The *POWER8 Functional Simulator Command Reference User's Manual* provides information about simulator commands that are available to configure and manage components in the simulation environment. These commands pertain to the command-line interface that is available after the simulator is installed and started.

Intended Audience

This document is intended for designers and programmers who are developing and testing applications that are designed to run on systems based on the POWER8 processor. Potential users include:

- System and software designers
- Hardware and software tool developers
- Application and product engineers

Using this Manual

This manual first presents preliminary topics that aid in understanding and using the POWER8 Functional Simulator commands and in managing the simulated machine. It then provides a command page for each available command.

Command Page Organization

Commands in this manual are arranged alphabetically. They typically contain the following information:

- **Command Name:** Provides a brief introduction to command functionality.
- **Tcl Syntax:** Specifies the full Tcl syntactic structure and grammar of the command statement. The syntax statements in this guide adhere to standard Tcl command notation, as described in *Conventions* on page 20. Some commands require one or more input parameters that must be passed to the command for it to execute successfully. In cases where multiple optional parameters are supported, the default is stated if applicable. For an explanation of how to read a synopsis statement, see *Section 1.4 Using the Command Pages* on page 26.
- **Description:** Describes the type of operation that is performed in the simulation by this command.
- **Arguments:** Describes each required or optional input argument.
- **Example Code and Output:** Provides sample code to demonstrate how the command is called, and displays corresponding output that is generated by the executed sample command sequence.

Conventions

The following typographical components are used to define special terms and command syntax:

Convention	Description
Bold typeface	Represents literal information, such as: <ul style="list-style-type: none"> Information and controls displayed on screen, including menu options, application pages, windows, dialogs, and field names. Commands, file names, and directories as used in general descriptions. In-line programming elements, such as function names and <u>XML</u> elements when referenced in the main text.
<i>Italics</i> typeface	Emphasizes new concepts and terms and to stress important ideas. In the case of command names, this font is used to denote user-specified components when describing command usage and functionality.
Bold monospaced typeface	Used in Tcl command format and syntax statements to denote the command name (as provided in each command page); for example, define config .
<i>Bold italics monospaced</i> typeface	Used in Tcl command format and syntax statements to denote the user-specified component in a command name (if applicable.) For example, in the <i>configuration_object</i> config command, the <i>configuration_object</i> represents the name of a configuration object on which the command action is performed.
<i>Italic monospaced</i> typeface	Delimits required parameters in Tcl command format and syntax statements or in sample code for which a value must be specified, such as in <code>cd /users/your_name</code> , where <i>your_name</i> denotes a user-specified input.
<i>?argument?</i> (Question-mark delimited)	Encloses optional parameters in format and syntax descriptions. For example, in the statement <i>machine bogus disk init devicenum imagepath accesstype ?cowpath? \$hash_size?</i> , the <i>?cowpath?</i> and <i>?hash_size?</i> parameters are optional.
Monospaced typeface	Used for example code, such as to represent Tcl or C/C++ code examples.
{ } (Braces)	Used in general descriptions to delimit a set of mutually exclusive user-specified parameter. Note: Braces are not used in Tcl command format and syntax statements, which follow the conventions defined previously.
(Vertical rule)	Separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press CTRL + ALT + DEL.
... (Horizontal or Vertical ellipsis)	In format and syntax descriptions, as well as in code examples, an ellipsis indicates that some material has been omitted to simplify a discussion.
Hyperlink	Web-based <u>URLs</u> are displayed in blue text to denote a virtual link to an external document. For example: http://www.ibm.com
Note: This is note text.	The note block denotes information that emphasizes a concept or provides critical information.
This is an inline footnote reference. ¹ 1. Descriptive footnote text.	A footnote annotates an explanatory note or reference inserted at the foot of the page that explains or expands upon a point within the text or indicates the source of a citation or peripheral information.

Related Documents

The following documents and links provide helpful information about Tcl/Tk used in the IBM® POWER8 Functional Simulator environment:

- *Practical Programming in Tcl and Tk* by Brent B. Welch. Prentice Hall, Inc.
- *Tcl/Tk in a Nutshell* by Paul Raines & Jeff Tranter, O'Reilly and Associates.

- The SourceForge.net Tcl Foundry located at <http://sourceforge.net/projects/tcl/>

Among the documents available in [OpenPOWER Connect](#), an IBM online technical library, the following are particularly helpful in understanding the operation of the POWER8 Functional Simulator:

- *POWER8 Processor User's Manual for the Single-Chip Module*
- *POWER8 Processor Datasheet for the Single-Chip Module*
- *POWER8 Processor SCM and Memory Buffer Hardware Errata Notice*
- *Power ISA User Instruction Set Architecture - Book I (Version 2.07)*
- *Power ISA Virtual Environment Architecture - Book II (Version 2.07)*
- *Power ISA Operating Environment Architecture (Server Environment) - Book III-S (Version 2.07)*

Help and Support

For questions or to request technical support:

1. Go to IBM [Customer Connect](https://www-03.ibm.com/technologyconnect/tgcm/login.jsp) (<https://www-03.ibm.com/technologyconnect/tgcm/login.jsp>).
2. Sign in with your IBM ID.
(New users must register first. Click "Registration" in the right navigation panel.)
3. Click on "Help and support" in the left navigation panel.
4. Scroll down to the section for POWER8 Functional Simulator Support.
5. Select either "Open a new Customer Connect support request" or "Manage existing Customer Connect support requests."

To provide additional feedback, contact OpenPOWER@us.ibm.com.



1. Introduction

This chapter describes the POWER8 Functional Simulator command framework and introduces the structure, format, and usage of simulator commands. Topics in this chapter include:

- Understanding and Using Simulator Commands
- Managing a Simulated Machine
- Overview of Command Structure and Syntax
- Using the Command Pages
- Accessing Help for Commands
- Top-Level Simulator Commands

1.1 Understanding and Using Simulator Commands

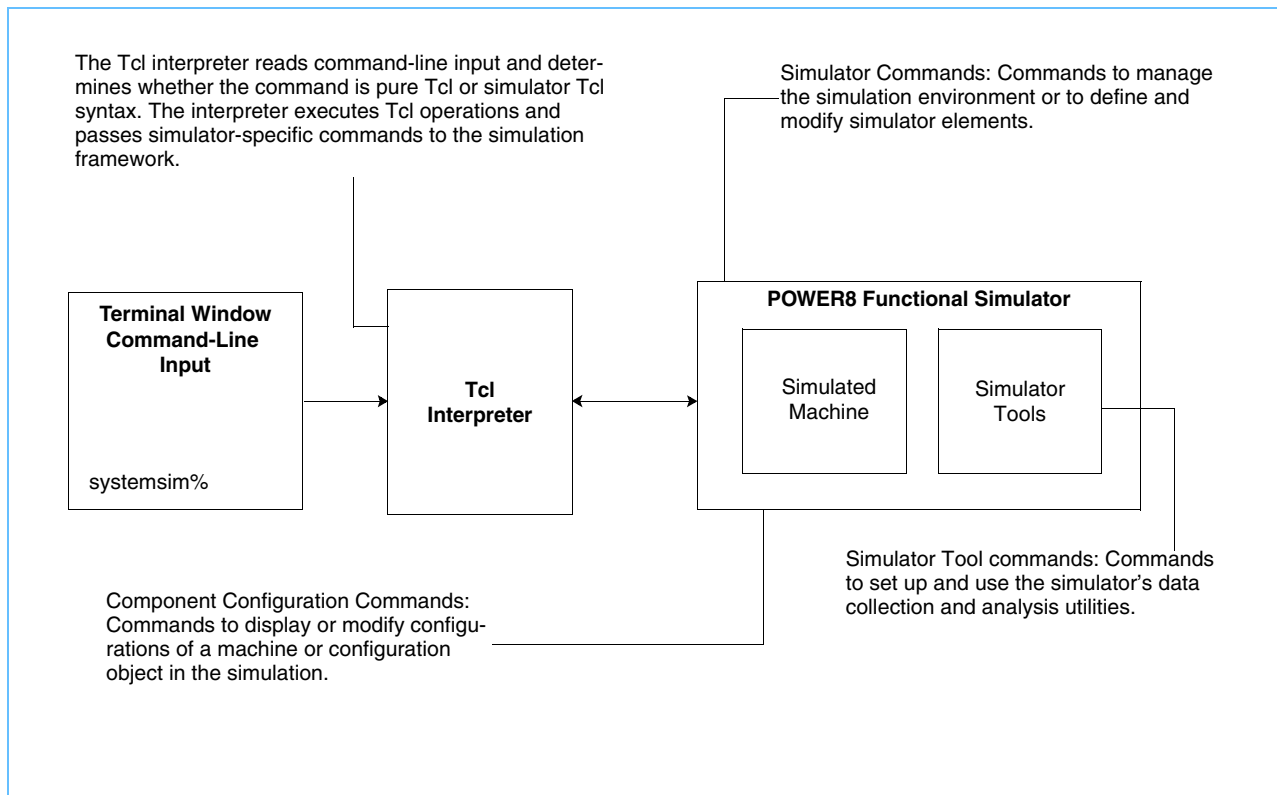
The POWER8 Functional Simulator uses Tcl/Tk to provide a simple and programmable command syntax that is easily extended and minimizes the need for proprietary programming grammar and usage. By extending Tcl with exported functions, data types, and numerous predefined interfaces for all inter-object communication, the simulator provides a rapid, cross-platform development environment that enables users to quickly start working in the simulation environment.

The POWER8 Functional Simulator command framework provides an extensive set of commands for modeling, simulating, and tuning microprocessor components in a system. Each component in a microprocessor system is configured through commands that not only define the component's run-time behavior and characteristics, but also govern its relationships and interactions with surrounding components in the system.

In addition to configuring system components, the simulator commands can be combined with programming logic and Tcl programming constructs to gather, analyze, and visualize simulation events, run workloads on the modeled microarchitecture, and generate performance metrics with new or revised configurations to forecast performance at future workloads. The command-line interface also can be used to perform a number of operations on the simulator itself, such as to control a simulation, start data collection, and define and load virtual devices and disk images.

Figure 1-1 on page 24 illustrates how commands are processed in the simulation environment and describes the different categories of commands that are available.

Figure 1-1. Categories of Simulator Commands



After the simulator is started, commands can be entered at the simulator command line or through simulation Tcl scripts. *Figure 1-2* on page 25 illustrates the simulator command line and the simulated Linux console that is launched from the simulator when the **boot-linux.tcl** script (found in the **run/pegasus/linux** directory) is executed by the **run/pegasus/power8.tcl** script.

Figure 1-2. Linux Boot of the POWER8 Functional Simulator with Console Display

```

File Edit View Search Terminal Help
.
3481601208: (1447295077): Debian GNU/Linux jessie/sid debian-ppc64el-2-0 hvc0
3481712651: (1447406520): debian-ppc64el-2-0 login: mambo^H ^H^H ^H^H ^H^H ^H^H
^Hroot
61036243565: (1663335773): Password:
66896767493: (1688554313): Last login: Thu Jun 13 22:29:04 UTC 2002 on hvc0
66936422093: (1728208942): Linux debian-ppc64el-2-0 3.16.3-systemsim #2 SMP Fri
Sep 26 13:56:21 UTC 2014 ppc64le

66936714824: (1728501673): The programs included with the Debian GNU/Linux syste
m are free software;
66936743456: (1728530305): the exact distribution terms for each program are des
cribed in the
66936769045: (1728555894): individual files in /usr/share/doc/*/copyright.

66936788746: (1728575595):
o the extent
66936817458: (1728604307):
66973648985: (1745055729):
s initialized

2068126571325: (8486412121)

```

DefaultConsole

```

[ OK ] Started Serial Getty on hvc0.
[ OK ] Reached target Login Prompts.
[FAILED] Failed to start Login Service.
See 'systemctl status systemd-logind.service' for details.
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.

Debian GNU/Linux jessie/sid debian-ppc64el-2-0 hvc0

debian-ppc64el-2-0 login: root
Password:
Last login: Thu Jun 13 22:29:04 UTC 2002 on hvc0
Linux debian-ppc64el-2-0 3.16.3-systemsim #2 SMP Fri Sep 26 13:56:21 UTC 2014 pp
c64le

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@debian-ppc64el-2-0:~# random: nonblocking pool is initialized
root@debian-ppc64el-2-0:~# █

```

1.2 Managing a Simulated Machine

Commands to set up and start a simulated machine can be entered at the simulator command line or through a Tcl configuration and start-up file that is loaded when the simulator starts. Each installation provides a default `lib/pegasus/systemsim.tcl` file that specifies a general set of commands to initialize the base simulation configurations and environment settings, create a local version of the simulator, and load and initialize one or more disk images. At startup, the POWER8 Functional Simulator interprets instructions in `lib/pegasus/systemsim.tcl` to set up default simulation behavior. Alternatively, a custom Tcl file is commonly used to start up and configure the simulation environment.

1.3 Overview of Command Structure and Syntax

Commands are organized into a hierarchy of operations based on the command function. At the top level, commands perform general sets of operations in the simulation environment, such as:

- Defining and displaying machine properties and system configurations
- Modifying configurable parameters
- Managing the simulation environment and its data collection and analysis tools.

The following command notation illustrates the general syntax structure of the simulator commands:

```
object [action] [{parameter1 . . . parameterN} [options]]
```

where:

- *object* is the entity on which the command action is performed. There are three types of objects to consider when specifying a command:
 - *Simulator.* Most commands are available to perform an action on the simulator itself. In this case, the simulator is the implied object of a command. It is not, therefore, explicitly stated in the command syntax. For example, the **quit** command simply exits the simulation environment and returns to the host command line. Likewise, the **version list** command determines and displays version and build information for the simulator.
 - *Machine or configuration objects.* After a machine configuration is created or a machine is instantiated, simulator commands can be used to manage these objects. For example, a system parameter can be reconfigured only after a configuration object, for example **myconf**, is created. Once the object is created, the **config** command is used to modify a setting in this configuration object. Configuration options might be limited for certain model builds.
 - *Utility objects.* The POWER8 Functional Simulator includes utilities for data collection and performance analysis that are called through commands, in which the utility is the object of the command. For example, the **simdebug** command allows you to list, set, and see status of which function-specific debug messages are enabled.
- *action* defines the type of operation that the command is performing on an object. Extending the **quit** command described earlier, **quit** stops the simulator and returns to the main shell. Likewise, the **display** command lists information about the simulation.
- *parameter1 . . . parameterN* specifies required input parameters. For example, the display command is a simulator command that can be used on a created machine configuration. To show the memory map of a created machine configuration named mysim, use the following command:

```
mysim display memorymap
```

- *options* lists arguments that modify the action that is performed on the object. Extending the previous simdebug example, you can list what debug message options are available and determine if they are active (1) or not (0) by executing the following command:

```
simdebug list
```

1.4 Using the Command Pages

Commands are arranged alphabetically. *Figure 1-3* on page 27 shows a sample syntax page that illustrates the format and general contents of command pages provided in this reference guide.

Figure 1-3. Sample Command Page Content and Format

<p>The command name at the top is followed by a short summary of the type of operation that is performed.</p>	<p>3.47 {machine} step The <i>{machine} step</i> command advances the simulator by a specified number of instructions.</p>
<p>Tcl Syntax specifies the grammar of a command. The syntactic structure includes any input arguments.</p>	<p>3.47.1 Tcl Syntax <i>machine step number_of_steps</i></p>
<p>Description provides more detailed information about how a command or command class is used in the simulation environment.</p>	<p>3.47.2 Description The <i>{machine} step</i> command advances the simulated machine by a specified number of instructions. Although this command is sometimes used for quickly forwarding the simulator to a specific point in the simulation, the <i>{machine} cycle</i> and <i>{machine} tick</i> commands are more commonly used to advance the system.</p>
<p>Arguments describe the parameters that are passed with the command.</p>	<p>3.47.3 Arguments <i>number_of_steps</i> Specifies the number of instructions to advance the machine.</p>
<p>Examples provides sample code to show how the command is called. If applicable, it displays corresponding output that is generated by the executed sample command sequence.</p>	<p>3.47.4 Examples Advance the simulator by 10000 steps: <pre>mysim step 10000</pre></p>
<p>Related Commands lists other commands that are relevant to the operation.</p>	<p>3.47.5 Related Commands</p> <ul style="list-style-type: none"> • <i>{machine} cycle</i> on page 73 • <i>{machine} tick</i> on page 108

1.5 Accessing Help for Commands

At any time, users can type the **help** command at the command line to retrieve a list of command choices that are available from that point in the syntax statement. In most cases, you can also just type a partial command sequence and press return. For example, at the top level, **help** displays a list of top-level commands. An arrow indicates that a subsequent level of command functionality is available for this command.

1.6 Top-Level Simulator Commands

Table 1-1 summarizes the functionality of selected top-level commands that are used to define, modify, and use the simulator. In the remainder of this chapter, command pages provide the complete command-line syntax and usage of each command or class of commands.

Table 1-1. POWER8 Functional Simulator Top-Level Commands (Page 1 of 2)

Command	Command Summary
alias	Assigns a user-specified personal shorthand for a command string. The alias command allows users to call a small, more familiar command or name to execute long or complex command strings.
define	Defines settings for a configuration object. The define command also provides that ability to duplicate configurations from a predefined machine type, instantiate a machine based on a configuration object, and enumerate a list of machines that are active in the simulation.
display	Displays system-wide information about configurations, machines, instruction settings, and warning levels. The display command is especially useful to determine properties that are configured for machines that are currently available in a simulation.
help or helprecursive	Displays a listing of simulator commands. The helprecursive command displays a comprehensive command tree that hierarchically lists syntax and input parameters for all available commands.

Table 1-1. POWER8 Functional Simulator Top-Level Commands (Page 2 of 2)

Command	Command Summary
modify	Modifies configurable simulation settings or parameters. The modify command is useful for changing various run-time parameters, such as the interval at which instructions are executed, the checkpoint type, latency cycles, or the warning level that is set for the simulation environment.
quit	Ends the current simulation and exits to the operating system command line.
simdebug	Provides low-level tracing capabilities that are useful for debugging functionality or performance issues in the simulated system.
simstop	Stops the simulation and waits for an instruction at the simulator command line. The simstop stop command stops the simulation, and the “simulation stopped (USER)” message is displayed. The simstop status command can be used in a tcl script to determine if the simulation has been stopped.
version	Displays the version number of simulation system components, the date and timestamp of the installed simulator build, and compile-time flags that are enabled in the build.

1.7 alias

The **alias** command enables users to call a small, more familiar command or name to execute long or complex command strings

1.7.1 Tcl Syntax

```
alias create alias_name cmdstring
alias create_unique cmdstring
alias delete alias_name
alias list
```

1.7.2 Description

The **alias** command assigns a user-specified shorthand for a command string. The alias command enables users to call a small, more familiar command or name to execute long or complex command strings.

alias create	Associates user-specified shorthand with a command string.
alias create_unique	Enables you to create a unique alias for an existing command in the format AL_0, AL_1, AL_2, and so on.
alias delete	Deletes an alias that was previously associated with a command string.
alias list	Generates a list of all the aliases and the command string that is associated with each.

1.7.3 Arguments

<i>alias_name</i>	Specifies the name of the alias that you are creating.
<i>command string</i>	Specifies the command string that you want to associate with the alias name.

1.7.4 Examples

1. Create an alias to display the contents of gpr8, instead of typing the entire command:

```
systemsim % mysim display gpr 8
0x0000000000000000

systemsim % alias create reg8 mysim display gpr 8
reg8

systemsim % reg8
0x0000000000000000
```

2. Create a unique alias:

```
systemsim % alias create_unique version
AL_0
systemsim % AL_0 list
systemsim % POWER8 Functional Simulator Version 1.0.2
```

```
CAPI Coherent Accelerator Processor Interface Version 0.11  
Built: 14:43:28 October 16 2014  
systemsim %
```

3. Generate a list of aliases:

```
systemsim % alias list  
reg8 {mysim display gpr 8}  
AL_0 {version}
```

1.8 define

The **define** command is used to define, configure, and instantiate a configuration object that is used to create a machine for the simulation environment. The various forms of this command are described in detail in *Section 2 Defining, Configuring, and Instantiating a Machine* on page 39.

For more information, see:

- *Section 2.1 define config* on page 40
- *Section 2.2 define cpu* on page 41
- *Section 2.3 define dup* on page 42
- *Section 2.4 define list* on page 43
- *Section 2.5 define machine* on page 44

1.9 display

The **display** command displays system-wide information about configurations, machines, instruction settings, and warning levels.

1.9.1 Tcl Syntax

```
display command_line_config_options
display command_line_user_options
display configures
display default_configure
display htm_status
display kips
display kips_dump_interval
display machines
display objects
display quiet_mode
display regress_mode
display standalone_mode
display warning
```

1.9.2 Description

The **display** commands show the current value of specific or selected options.

<i>display command_line_config_options</i>	Shows the command-line configuration options.
<i>display command_line_user_options</i>	Shows the command-line user options.
<i>display configures</i>	Shows the existing, defined configurations available for use.
<i>display default_configure</i>	Shows the default configuration that is used if no other is specified.
<i>display htm_status</i>	Shows the status of transactional memory (HTM).
<i>display kips</i>	Shows the default instructions per second, in thousands (kips) for this model.
<i>display kips_dump_interval</i>	Shows a parameter that is not applicable for this model; do not alter from the default value.
<i>display machines</i>	Shows the machines that are currently defined.
<i>display quiet_mode</i>	Shows the on or off status of quiet mode.
<i>display regress_mode</i>	Shows the on or off status of regress mode.
<i>display standalone_mode</i>	Shows the on or off status of standalone mode.
<i>display warning</i>	Shows the current warning level.

1.9.3 Examples

1. Display the current warning level:

```
systemsim % display warning
```


The following output is displayed:

```
Warning level is 3
```

2. Display the status of quiet mode:

```
systemsim % display quiet_mode
```

The following output is displayed:

```
off
```

1.10 help or helprecursive

The **help** and **helprecursive** commands provide a built-in help function.

1.10.1 Tcl Syntax

```
help
command line command help
helprecursive
```

1.10.2 Description

The **help** and **helprecursive** commands provide context-sensitive help.

help When entered at the command-line prompt, **help** provides top-level help. That is, it displays a list of the next level of commands. Commands that are followed by an arrow (->) symbol are commands that **helprecursive** can expand further.

When entered after a simulator command, **help** provides hints about how to correctly form that command. That is, it indicates that correct syntax for the command.

helprecursive When entered after a simulator command, **helprecursive** provides a list of all the commands and subcommands available for a particular command. To see all commands and subcommands available for **mysim**, type

```
mysim helprecursive
```

1.10.3 Examples

mysim thread help: Available Commands

```
config_on
dtranslate
itranslate
setargs [args list]
stall
start_thread {PC_address}
step {number of instructions}
stop_thread
cpu {cpu-number} ->
display ->
interrupt ->
load ->
mcm {mcm-number} ->
memory ->
osinfo ->
process ->
set ->
thread {thread-number} ->
util ->
```

1.11 quit

The **quit** command ends the current simulation and exits to the operating-system command line.

1.11.1 Tcl Syntax

quit

1.11.2 Description

The **quit** command exits the simulator.

1.11.3 Examples

quit

1.12 simdebug

The **simdebug** command provides low-level tracing capabilities that are useful for debugging functionality or performance issues in the simulated system.

1.12.1 Tcl Syntax

```
simdebug list
simdebug set name value
simdebug status name
```

1.12.2 Description

The **simdebug** command controls the debug output generated by the simulator or lists the various debug types and their output configuration. A one indicates on or enabled; a zero indicates off or disabled.

1.12.3 Arguments

<i>name</i>	Name of the output control parameter you are modifying.
<i>value</i>	Either 1 (enabled/on) or 0 (disabled/off).

1.12.4 Examples

Display the status of the simdebug named “helpful:”

```
systemsim % simdebug status helpful
```

The following output is displayed:

```
1
```

1.13 simstop

The **simstop** command stops the simulation and waits for an instruction at the simulator command line. The **simstop** command performs the same operation as typing **CTRL+C** to interrupt the simulation.

1.13.1 Tcl Syntax

```
simstop status
simstop stop
```

1.13.2 Description

The **simstop** command can be executed from a Tcl script to cause the simulation to stop. The simulation can then be restarted from the simulator command line. This allows users to run a script to a certain point, then do manual command execution.

1.13.3 Examples

```
puts "console line: $triginfo(linenum): string trim $triginfo(line) "\r\n"]"
puts "Stopping"
simstop
```

1.14 version

The **version** command displays the version number and timestamp details of simulation system components and build.

1.14.1 Tcl Syntax

```
version display  
version flags  
version list
```

1.14.2 Description

Displays the version number of simulation system components, the date and timestamp of the installed simulator build, and compile-time flags that are enabled in the build.

1.14.3 Examples

Display the version number of the simulation system components and the date and timestamps:

```
systemsim % version list
```

The following output is displayed:

```
POWER8 Functional Simulator Version 1.0.2  
CAPI Coherent Accelerator Processor Interface Version 0.11  
OpenSSL_Built OpenSSL 0.9.8e-rhel5 01 Jul 2008  
Built: 12:28:26 Sep 25 2014
```

2. Defining, Configuring, and Instantiating a Machine

This chapter describes commands that are used to define and configure a configuration object that is used to create a machine for the simulation environment. *Section 1 Introduction* on page 23 presents concepts that are helpful in understanding how to configure and use a simulated machine in the simulator environment. *Section 1.2 Managing a Simulated Machine* on page 25 illustrates the process by which a configuration object is defined, configured, and used to instantiate the simulated machine.

2.1 define config

The **define config** command defines a new configuration object.

2.1.1 Tcl Syntax

```
define config new_configuration_object
```

2.1.2 Description

The **define config** command creates a new configuration object. A configuration object is a named collection, such as **myconf**, of machine properties that are used to instantiate a machine that is run in the simulator. The **define config** command creates and populates a new configuration object with default properties that have been defined for a machine type. Once an object is defined, the default properties can be reconfigured with the *{configuration_object}* **config** command before the **define machine** command is used to instantiate a machine object from this configuration.

2.1.3 Arguments

new_configuration_object Specifies the name of the new configuration object.

2.1.4 Examples

```
define config myconf
```

where **myconf** is a configuration object that defines baseline settings for a machine that can be run in a simulation.

2.1.5 Related Commands

- **define cpu** on page 41
- **define dup** on page 42
- **define machine** on page 44
- **display configures** on page 46

2.2 define cpu

The **define cpu** command defines a new CPU.

2.2.1 Tcl Syntax

```
define cpu config-name cpu-name
```

2.2.2 Description

The **define cpu** command defines a new CPU.

2.2.3 Arguments

config-name

Specifies the name of the configuration object.

cpu-name

The CPU name (cpu 0, cpu 1, and so on) you choose to associate with your configuration.

2.2.4 Examples

```
define config myconf
```

where **myconf** is a configuration object that defines baseline settings for a machine that can be run in a simulation.

2.2.5 Related Commands

- **define config** on page 40
- **define dup** on page 42
- **define machine** on page 44
- **display configures** on page 46

2.3 define dup

The **define dup** command defines a duplicate configuration object based on an existing machine type.

2.3.1 Tcl Syntax

```
define dup existing_configuration_object new_configuration_object
```

2.3.2 Description

The **define dup** command duplicates a *configuration object* from an existing configuration. A configuration object is a named collection, such as **myconf**, of machine properties that are used to instantiate a machine that is run in the simulator. Although the **define dup** command provides similar functionality as the **define config** command, unlike **define config** it requires an existing configuration object and cannot be used to create a configuration object with different machine properties.

2.3.3 Arguments

<i>existing_configuration_object</i>	Specifies the name of an existing configuration object that will be duplicated.
<i>new_configuration_object</i>	Specifies the name of the duplicate configuration object to be created.

2.3.4 Examples

1. Define a **myconf** configuration object into which all properties and values of the pre-defined **P8** configuration object for the POWER8 processor are duplicated. This **myconf** object can then be used to modify one or more configurations and instantiate a custom POWER8 machine based on the revised settings:

```
define dup P8 myconf
```

2. Define an **anotherconf** configuration object to duplicate the **myconf** configuration object created in step 1:

```
define dup myconf anotherconf
```

2.3.5 Related Commands

- **define config** on page 40
- **define cpu** on page 41
- **define machine** on page 44
- **display configures** on page 46

2.4 define list

The **define list** command lists all machines that are active in a simulation.

2.4.1 Tcl Syntax

```
define list
```

2.4.2 Description

The **define list** command provides a list of machines that are active in the simulation environment.

2.4.3 Examples

Output the name of all existing machines that have been created from configurations. In the case where the **mysim** and **anothersim** machines are defined, typing the **define list** command at the simulator prompt displays the following output:

```
anothersim mysim
```

2.4.4 Related Commands

- **define config** on page 40
- **define cpu** on page 41
- **define dup** on page 42
- **define machine** on page 44

2.5 define machine

The **define machine** command instantiates a machine from a configuration object.

2.5.1 Tcl Syntax

```
define machine configuration_object_name machine_name
```

2.5.2 Description

The **define machine** command instantiates a machine that is used in a simulation. Multiple machines can be created in the simulator based on a single configuration object.

2.5.3 Arguments

<i>configuration_object_name</i>	Specifies the name of an existing configuration object that will be duplicated.
<i>machine_name</i>	Specifies the name of the duplicate configuration object to be created.

2.5.4 Examples

Instantiate a **mysim** machine from the **myconf** configuration object:

```
define machine myconf mysim
```

Once created, the **mysim** machine can be run in a simulation to capture details about the functionality and performance of this machine configuration.

2.5.5 Related Commands

- **define config** on page 40
- **define cpu** on page 41
- **define dup** on page 42
- **define list** on page 43

2.6 {configuration_object} config

The *{configuration_object}* **config** command assigns a value to a configuration property.

2.6.1 Tcl Syntax

```
configuration_object config property_name property_value
```

2.6.2 Description

The *{configuration_object}* **config** command assigns a new value to an existing property in a configuration object.

2.6.3 Arguments

<i>property_name</i>	Specifies the name of an existing configuration property whose current value is replaced with the specified value. See {configuration_object} display on page 46, which can be used to display a complete list of properties.
<i>property_value</i>	Specifies the value that is assigned to the configuration property.

2.6.4 Examples

1. Reconfigure the default *memory_size* property in the **myconf** configuration object to be 128 MB:

```
myconf config memory_size 128M
```

Note: For properties that specify a size parameter, such as *memory_size*, the actual size of the value must be provided. For example, the sample code above sets the machine memory size to 128M, which the simulator interprets as a scaled integer and sets the memory size to 128 *megabytes*. If the value 128 is specified, the memory size is set to 128 *bytes*.

2. Verify that the *memory_size* parameter for **myconf** has been modified:

```
myconf display memory_size
```

The following output is displayed:

```
myconf:
  memory_size = 128M (number of bytes of memory)
```

2.6.5 Related Commands

- **{configuration_object} display** on page 46
- **define config** on page 40

2.7 {configuration_object} display

The {*configuration_object*} **display** command displays a list of configuration properties.

2.7.1 Tcl Syntax

```
configuration_object display $property_name  
configuration_object display command_line_config_options  
configuration_object display command_line_user_options  
configuration_object display configures  
configuration_object display default_configure  
configuration_object display htm_status  
configuration_object display itimes [symbolic command]  
configuration_object display kips  
configuration_object display kips_dump_interval  
configuration_object display machines  
configuration_object display objects  
configuration_object display quiet_mode  
configuration_object display regress_mode  
configuration_object display standalone_mode  
configuration_object display warning
```

2.7.2 Description

The {*configuration_object*} **display** command outputs the name and value of all properties that correspond to the specified search string for the given configuration object. If a property name is not specified, the entire list of properties for the configuration object is displayed. A valid search string can include any sequence of characters, based on which the simulator returns information about all properties that contain the given string.

2.7.3 Arguments

property_name (Optional) Specifies the name of an existing configuration property that is displayed to standard output. If *property_name* is not specified, the entire list of properties for the configuration object is displayed.

2.7.4 Examples

1. View all configuration properties containing the string, memory:

```
myconf display memory
```

The following output is displayed:

```
myconf:  
  memory_size = 64M (number of bytes of memory)  
  memory_start = 0 (address of first byte of memory)
```

2. View all configuration properties containing the string, M:

```
myconf display M
```

The following output is displayed:

myconf:

```
  htm/htm_os_supports_tm = FALSE (does the OS support TM operations)
    machine_type = pegasus (type of machine: WALNUT, SPRUCE, ...)
    memory_size = 64M (number of bytes of memory)
    memory_start = 0 (address of first byte of memory)
```

2.7.5 Related Commands

- `{configuration_object} config` on page 45

2.8 {configuration_object} exit

The *{configuration_object}* **exit** command deletes a configuration object.

2.8.1 Tcl Syntax

configuration_object **exit**

2.8.2 Description

The *{configuration_object}* **exit** command removes the specified configuration object from the simulation environment. The *{configuration_object}* **exit** command performs the same operation as the *{configuration_object}* **quit** command.

2.8.3 Examples

1. Delete the **tmpconf** configuration object from the simulation environment:

```
tmpconf exit
```

Note: The *{configuration_object}* **exit** command permanently removes the object from the system. Before removing a configuration object, ensure that you do not need the configurations specified in this object.

2. Verify that the **tmpconf** object has been removed by listing all active configuration objects and machines in the system:

```
display configures
```

2.8.4 Related Commands

- **{configuration_object} display** on page 46
- **{configuration_object} quit** on page 50
- **display configures** on page 46

2.9 {configuration_object} query

The *{configuration_object}* **query** command displays the value of a configuration object property.

2.9.1 Tcl Syntax

```
configuration_object query full_property_name
```

2.9.2 Description

The *{configuration_object}* **query** command returns the value of the specified property for the given configuration object. Although the *{configuration_object}* **display** command provides similar functionality as *{configuration_object}* **query**, the return value from *{configuration_object}* **query** is formatted such that it can be conveniently used as input in a Tcl procedure.

2.9.3 Arguments

full_property_name Specifies the complete name of an existing configuration property.

2.9.4 Examples

View the value assigned to the **memory_size** configuration property:

```
myconf query memory_size
```

The following output is displayed:

```
64M
```

Alternatively, using the *{configuration_object}* **display** command to view memory size results in the following output:

```
myconf:  
  memory_size = 64M (number of bytes of memory)
```

2.9.5 Related Commands

- **{configuration_object} display** on page 46

2.10 {configuration_object} quit

The *{configuration_object}* **quit** command deletes a configuration object.

2.10.1 Tcl Syntax

configuration_object quit

2.10.2 Description

The *{configuration_object}* *quit* command removes the specified configuration object from the simulation environment. The *{configuration_object}* **quit** command performs the same operation as the *{configuration_object}* **exit** command.

2.10.3 Examples

1. Delete the **tmpconf** configuration object from the simulation environment:

```
tmpconf quit
```

Note: The *{configuration_object}* **quit** command permanently removes the object from the system. Before removing a configuration object, ensure that you do not need the configurations specified in this object.

2. Verify that the **tmpconf** object has been removed by listing all active configuration objects and machines in the system:

```
display configures
```

2.10.4 Related Commands

- **{configuration_object} display** on page 46
- **{configuration_object} exit** on page 48
- **display configures** on page 46



3. Configuring and Modifying Machine Properties

This chapter describes commands that are used to configure and modify machine properties in the simulation environment.

3.1 {machine} bogus disk cleanup

The {machine} **bogus disk cleanup** command removes a bogus disk from the system.

3.1.1 Tcl Syntax

machine bogus disk cleanup

3.1.2 Description

Bogus disk support provides a high-performance call-through interface to block devices, disk images, or both that reside on the host system. The {machine} **bogus disk cleanup** command extends functionality provided by the {machine} **bogus disk commands** by synchronizing all data and cleaning up all references to virtual disk device images that have been loaded in the simulation. Although the **bogus disk** commands integrate this functionality into their exit routines, issuing the {machine} **bogus disk cleanup** command before exiting a simulation ensures that files are properly synchronized.

3.1.3 Examples

Issue the {machine} **bogus disk cleanup** command to synchronize disk images in the system:

```
mysim bogus disk cleanup
```

The simulator prints a confirmation message:

```
ok
```

3.1.4 Related Commands

- {machine} **bogus disk init** on page 54
- {machine} **bogus disk sync** on page 57

3.2 {machine} bogus disk display

The {machine} **bogus disk display** command shows the minor number assigned to the bogus device by the simulated system, the path to the device on the host system, the block and device sizes configured for the bogus disk, and the access type (read, write, or copy on write).

3.2.1 Tcl Syntax

machine bogus disk display

3.2.2 Description

The {machine} **bogus disk display** command is only useful when there is an active bogus disk that has already been initialized and mounted. Then it can be used to display information about the active bogus disk.

3.2.3 Examples

Issue the {*machine*} **bogus disk display** command to display information about the active bogus disk:

```
systemsim % mysim bogus disk display
```

The simulator displays the statistics:

```
{ minor 0 path /home/mambo_kernels/debian-ppc64le-rootfs-v2.0.img blksize 4096 devsize 10485760  
cow_flags rw cow_path (null) }
```

3.2.4 Related Commands

- {machine} **bogus disk init** on page 54
- {machine} **bogus disk sync** on page 57

3.3 {machine} bogus disk init

The *{machine}* **bogus disk init** command loads and initializes a virtual disk device into simulation.

3.3.1 Tcl Syntax

```
machine bogus disk init devicenum imagepath accesstype ?cowpath? $hash_size?
```

3.3.2 Description

Bogus disk support provides a high-performance call-through interface to block devices, disk images, or both that reside on the host system. The *{machine}* **bogus disk init** command loads a disk image residing on the host machine into the simulation environment. This command interface serves as the BIOS of a regular machine that links a specific hard drive/rootdisk to an internal device number.

Before the newly instantiated machine can run in simulation, a rootdisk image must be loaded in the system, which contains the root environment for the system. This image provides a snapshot of a functioning Linux system that is available inside the simulator, including all tools, libraries, and debuggers that are required to run an actual Linux system. This enables the simulator to provide the appropriate run-time support to run applications as they are executed in an actual Linux environment. The simulator is designed to optimize the execution of the Linux kernel by reading contents of the rootdisk image as the simulator traverses the root file system in the simulator environment.

3.3.3 Arguments

<i>devicenum</i>	Specifies a device number (starting at 0) that correlates to a device in Linux.
<i>imagepath</i>	Specifies the full path name to the disk image.
<i>accesstype</i>	Specifies the access type to the disk file. Supported access types include read (r), write only (w), read-write (rw), copy-on-write (cow), or an alternate copy-on-write (newcow) method. The cow access type is an optimization strategy that specifies that all write operations are performed on a snapshot of the original disk image so that the state of the original image is maintained. If the simulation fails or is stopped, it can be resumed from either the point at which it stopped, or it can revert to the original disk image. The newcow access type provides another type of the copy-on-write functionality in which a new version of the disk image is created each time the simulation starts.
<i>cowpath</i>	Specifies the file path of the snapshot disk image. ¹
<i>hashsize</i>	Specifies the size of the hash table that is storing the snapshot images. The hash table size is a performance-to-memory trade-off.

1. A **cowpath.table** is an internal hash table that is used to map sectors in the original disk image to sectors in the snapshot image file. When a disk image is loaded for the first time with the **cow** access method, the snapshot disk file and **cowpath.table** file are automatically created. If these files already exist, however, the mapping in the **cowpath.table** is used to load the current state of the disk image file. In essence, all changes from the previous session are loaded and the POWER8 Functional Simulator can continue from that point.

3.3.4 Examples

The following sample lines of code are added in the **.boot-linux.tcl** configuration and startup file to mount the Linux kernel and rootdisk images. As illustrated in this code, the simulator provides distinct commands for loading the Linux kernel (line 53) versus loading the rootdisk image (line 42).

```
41 #bogus disk
42 mysim bogus disk init 0 disk.img rw
.
.
.
52 # Load vmlinux
53 mysim load vmlinux vmlinux 0
```

The following *{machine}* **bogus disk** command creates a copy-on-write disk image located in **/tmp/hdisk0cow** and a copy-on-write table at **/tmp/hdisk0cow.table**. The internal hash table size is set to 1024. The hash-table size is a performance-to-memory trade-off. Increasing the hash-table size improves performance in an environment where many changes are made.

```
mysim bogus disk init 0 /tmp/hdisk0 cow /tmp/hdisk0cow 1024
```

3.3.5 Related Commands

- *{machine}* **bogus disk sync** on page 57
- *{machine}* **load vmlinux** on page 95

3.4 {machine} bogus disk stat

The {machine} **bogus disk stat** command shows the minor number assigned to the bogus device by the simulated system, the path to the device on the host system, and the read/write event statistics.

3.4.1 Tcl Syntax

machine bogus disk stat

3.4.2 Description

The {machine} **bogus disk stat** command is only useful when there is an active bogus disk that has already been initialized and mounted. Then it can be used to show statistics.

3.4.3 Examples

Issue the {*machine*} **bogus disk stat** command to display statistics:

```
systemsim % mysim bogus disk stat
```

The simulator displays the statistics:

```
{ minor 0 path /home/mambo_kernels/debian-ppc64le-rootfs-v2.0.img reads 24574976 writes  
1622016 }
```

3.4.4 Related Commands

- {machine} **bogus disk init** on page 54
- {machine} **bogus disk sync** on page 57

3.5 {*machine*} bogus disk sync

The *{machine}* **bogus disk sync** command synchronizes a copy-on-write file for subsequent usage.

3.5.1 Tcl Syntax

machine **bogus disk sync** *devicenum*

3.5.2 Description

The *{machine}* **bogus disk sync** command maintains the integrity of an existing copy-on-write image file so that it is usable in subsequent sessions. Although the **bogus disk** commands integrate this functionality into their exit routines, issuing this command before exiting a simulation ensures that the copy-on-write file is properly synchronized.

3.5.3 Arguments

devicenum Specifies a device number used when the disk was initialized.

3.5.4 Examples

The following *{machine}* **bogus disk sync** command synchronizes the **mambobd0** disk image and writes the copy-on-write image and table files:

```
mysim bogus disk sync 0
```

3.5.5 Related Commands

- **{machine} bogus disk init** on page 54

3.6 {machine} bogus net cleanup

The {machine} **bogus net cleanup** command removes a bogus network device connection from the system.

3.6.1 Tcl Syntax

machine bogus net cleanup

3.6.2 Description

Bogus net support provides a high-performance call-through interface to simulate Ethernet connectivity between a simulated Ethernet adapter and the Ethernet adapters that reside on the host system. The {*machine*} **bogus net cleanup** command extends functionality provided by {*machine*} **bogus net** by cleaning up all references to virtual net devices that have been initialized and are in use in the simulation. Although the **bogus net init** command integrates this functionality into its exit routines, issuing the {*machine*} **bogus net cleanup** command before exiting a simulation ensures that simulated network device closure is done correctly.

3.6.3 Examples

Issue the {*machine*} **bogus net cleanup** command to correctly close the network devices configured in the system:

```
mysim bogus net cleanup
```

The simulator prints a confirmation message:

```
ok
```

3.6.4 Related Commands

- {*machine*} **bogus net init** on page 59

3.7 {machine} bogus net init

The *{machine}* **bogus net init** command loads and initializes a virtual Ethernet device.

3.7.1 Tcl Syntax

```
machine bogus net init devicenum MACaddr socket_file IRQ IRQ_offset
```

3.7.2 Description

The *{machine}* **bogus net init** command supports a call-through interface to an Ethernet network. This interface is provided through an emulated Ethernet device (**mambonet0**) and a separate utility that interfaces with a network device on the host system.

3.7.2.1 Extended Description of Bogus Network Support

There are three key components to bogus network communications:

1. A facility on the host system that provides systemsim-p8 with a path to the network. The TUN/TAP support available for Linux is a good choice for this component. TUN/TAP is assumed in the remainder of this description.
2. The systemsim-p8 support for the bogus network. This support is not enabled by default. Simulator commands are used to enable the bogus network support.
3. An operating system (OS) kernel with a bogus network driver.

3.7.2.2 Setting up TUN/TAP on the Host System

You must have root privileges on your system to set up bogus network operation. Execute the following commands:

```
sudo tunc1l -u $USER -t tap0
sudo ifconfig tap0 172.19.98.108 netmask 255.255.255.254
```

3.7.2.3 Configuring systemsim-p8 Support for the Bogus Network

To enable bogus network support, issue simulator commands that configure and initialize the bogus network. These commands must be issued before booting the Linux kernel on the simulator so that Linux recognizes the bogus network device during its boot process. The general form of the command to initialize the bogus network is:

```
mysim bogus net init 0 <mac address> <interface name> <irq>
```

The *<mac address>* parameter is the media access control (MAC) hardware address that you want the emulated Ethernet to use. It must be unique on your network (that is, not used by any other emulated hosts or by any host network adapter). The *<interface name>* parameter is the name of the interface to be used, typically "tap0." The *<irq>* parameter specifies the interrupt request queue ID to be used by the bogus network device; use 0 0 for the POWER8 Functional Simulator.

3.7.3 Arguments

<i>devicenum</i>	Specifies a device number (starting at 0) that correlates to a device in Linux.
<i>MACaddr</i>	Specifies the <u>MAC</u> hardware address that the emulated Ethernet will use. This must be a unique address on the network (that is, one that is not used by any other emulated hosts or by any host network adapter).
<i>interface name</i>	The <i>interface name</i> parameter is the name of the interface to be used, typically "tap0."
<i>IRQ</i> and <i>IRQ_offset</i>	The <i>IRQ</i> and <i>IRQ_offset</i> parameters specify the interrupt request queue ID to be used by the bogus network device; use 0 0 for the POWER8 Functional Simulator. ¹

1. If an interrupt and interrupt offset are not specified, the polled mode is used by default.

3.7.4 Examples

The following *{machine}* **bogus net init** command interfaces with the **eth0** network device using the **de:ad:ca:fe:ba:be** MAC address where the default **netapp** socket and IRQ offset are used:

```
mysim bogus net init 0 de:ad:ca:fe:ba:be tap0 0 0
```

3.7.5 Related Commands

- *{machine}* **bogus disk init** on page 54
- *{machine}* **bogus net cleanup** on page 58

3.8 {machine} bogushalt

The *{machine}* **bogushalt** command suspends simulator activity for a specified amount of time.

3.8.1 Tcl Syntax

```
machine bogushalt delay delay_in_ticks  
machine bogushalt disable  
machine bogushalt display  
machine bogushalt enable
```

3.8.2 Description

The *{machine}* **bogushalt** command simulates an IBM POWER® halt instruction. When the simulator encounters a *{machine}* **bogushalt delay**, it suspends activity for the specified number of seconds and waits for another instruction. The following subcommands are available in the *{machine}* **bogushalt** set of commands:

<i>{machine}</i> bogushalt delay <i>{integer}</i>	Specifies a halt instruction with the specified delay.
<i>{machine}</i> bogushalt disable	Disables the <i>{machine}</i> bogushalt delay command.
<i>{machine}</i> bogushalt display	Prints the current status of the <i>{machine}</i> bogushalt command.
<i>{machine}</i> bogushalt enable	Enables the <i>{machine}</i> bogushalt delay command.

3.8.3 Arguments

delay_in_ticks Specifies an integer amount for the number of ticks that the simulator will delay processing instructions.

3.8.4 Examples

Set a delay of 120 ticks before reading the next instruction:

```
mysim bogushalt delay 120
```

The following output is displayed:

```
Bogus halt delay is set to 120 ticks
```

3.8.5 Related Commands

- *{machine}* **bogus disk init** on page 54

3.9 {machine} capi

The *{machine}* **capi** commands are used for the Coherent Accelerator Processor Interface (CAPI) model.

3.9.1 Tcl Syntax

```

machine capi attach type slice freq (not supported in this release)
machine capi connect slice freq server_host port
machine capi create p1_base p2_base slices
machine capi debug level (not supported in this release)
machine capi display attribute (not supported in this release)
machine capi dtranslate slice ea (not supported in this release)
machine capi dump slice (not supported in this release)
machine capi enable feature on/off (not supported in this release)

```

3.9.2 Description

The *{machine}* **capi** commands are used to create the power service layer (PSL), connect to a socket (for co-sim), and control and display attributes of the CAPI model.

3.9.3 Arguments

<i>attribute</i>	Defines the attribute of the current value that you want to display. Not currently supported for the capi command.
<i>ea</i>	The effective address used by the capi dtranslate command, which translates the effective address given to a real address as a data reference. Not currently supported for the capi command.
<i>feature on/off</i>	Not currently supported for the capi command.
<i>freq</i>	Suggested value for 2 GHZ operation of simulated CPU is 2000000000.
<i>level</i>	Not currently supported for the capi command.
<i>p1_base</i>	Hexadecimal value for the base address of the P1 area in system memory (see the <i>Coherent Accelerator Interface Architecture</i> specification for details).
<i>p2_base</i>	Hexadecimal value for the base address of the P2 area in system memory (for more information, see the <i>Coherent Accelerator Interface Architecture specification</i> .
<i>port</i>	Socket port to connect to for communication with the accelerator function unit (AFU). The default is 32769.
<i>server_host</i>	The default is localhost but can be any IP address that the host system can connect with.
<i>slice</i>	The only accepted value is 0.
<i>slices</i>	Number of AFUs per PSL (the only supported value is 1).

3.9.4 Examples

In a machine with a single coherent accelerator processor proxy (CAPP), create the PSL model and connect to an AFU using a socket (port 32769) on the localhost:

```
mysim capi create 0x0000040000000000 0x0000050000000000 1
mysim capi connect 0 1600000 localhost 32769
```

3.10 {machine} config_on

The *{machine}* **config_on** command starts the processor component.

3.10.1 Tcl Syntax

machine config_on

3.10.2 Description

The *{machine}* **config_on** command performs the same operation as the *{machine}* **cpu 0 config_on** command (see [Section 3.18 {machine} cpu](#) on page 73) to control the state of the default processor (CPU 0). In hardware, this operation is generally implemented by moving a mode bit in the processor's Machine State Register (MSR).

When a machine is created in the simulator, the processor is set to off by default. In a typical environment, the processor is turned on as a side effect of loading the machine, which initializes the machine state. The *{machine}* **config_on** command is necessary only when the machine is state is not defined by any initial processing.

3.10.3 Examples

In a machine with a single processor, start the processor:

```
mysim config_on
```

The following message is displayed to confirm that the processor is started:

```
CPU 0 set running
```

3.10.4 Related Commands

- **{machine} cpu** on page 73
- **{machine} stall** on page 107

3.11 {machine} console create

The *{machine}* **console create** command defines the source (or sink) of characters for the simulation console.

3.11.1 Tcl Syntax

```
machine console create console_id in|inout|out file file_name
machine console create console_id in|inout|out listen port
machine console create console_id in|inout|out program program_name
machine console create console_id in|inout|out socket host:port
machine console create console_id in|inout|out string input_string
```

3.11.2 Description

The *{machine}* **console create** command feeds characters from the specified source into the simulation and vice versa. The simulated machine is designed to accept input from the user, which normally is typed directly into the simulator console window. However, in the case of predefined scripts, the input can be obtained from additional sources, such as text files or programs. The input characters are copied to the simulation as if they are typed into the console interface. The simulation environment provides different *{machine}* **console create** commands, each of which defines the source (or sink) of the characters for the console. These command options are listed in the Arguments section.

3.11.3 Arguments

<i>console_id</i>	Specifies a user-defined console name. This identifier is a unique name associated with a console. The <i>console_id</i> enables the console that is being created to be identified in a list and is used with other console-related commands, such as enabling, disabling, or deleting a console.
<i>in inout out</i>	Specifies whether the console will be used for input (in), output (out), or input and output (inout). Input consoles provide characters that are given to the machine, and output consoles display any characters that the computer generates.
file <i>filename</i>	Indicates that the specified file is used to read input or record output results, or both.
listen <i>port</i>	Indicates that the simulator will create the specified port and then listen for another program to attach to this port.
program <i>prog_name</i>	Indicates that characters will be sent to or from a specified program. For example, an Xterm window is an instance of the program option, since it is normally the display for the machine console. When the program option is used, communication occurs over pseudo-terminals. The pseudo-TTYS provide interfaces to receive nonblocking input as well as to send terminal control characters.
socket <i>host:port</i>	Indicates that the simulator will attempt to connect to the named host and port.
string <i>input_string</i>	Indicates that the source input is given in the string command. Strings cannot be used for output consoles.

3.11.4 Examples

The following sample lines of code are added in the `.systemsim.tcl` configuration and startup file to manage the simulation from the console window. The console `callthru exit` command (line 17) exits the POWER8 Functional Simulator console. A simple `GCC` compile operation on source that is called from the host to the simulated system (lines 22 – 25) is executed. The simulation is then started.

```
.
.
.
16: # exit console to start simulation
17: mysim console create input in string "callthru exit"
.
.
.
21: # invoke call-thrus to start instructions to compile C program
22: mysim console create input in string "callthru source hello.c >
hello.c"
23: mysim console create input in string "gcc -o hello hello.c"
24: mysim console create input in string "./hello"
25: mysim console create input in string "callthru exit"
```

3.11.5 Related Commands

- `{machine} console create` on page 65
- `{machine} console destroy` on page 67

3.12 {machine} console destroy

The *{machine}* **console destroy** command removes the specified console.

3.12.1 Tcl Syntax

```
machine console destroy console_id
```

3.12.2 Description

The *{machine}* **console destroy** command removes the specified console from its use in input and output, and deletes all its supporting data structures. An EOF is generated for any output file or socket.

3.12.3 Arguments

console_id Specifies the name of the console to be removed.

3.12.4 Examples

The following sample command line removes the c0 console from the simulation:

```
mysim console destroy c0
```

3.12.5 Related Commands

- **{machine} console create** on page 65
- **{machine} console disable** on page 68
- **{machine} console enable** on page 70
- **{machine} console list** on page 71

3.13 {machine} console disable

The *{machine}* **console disable** command disables a simulation console.

3.13.1 Tcl Syntax

```
machine console disable console_id
```

3.13.2 Description

Once created, an input console can be disabled so that it cannot read input characters; likewise, a disabled output console does not generate output characters from the machine. The *{machine}* **console disable** command suspends the specified console from its use in input and output.

3.13.3 Arguments

console_id Specifies the name of the console to be disabled.

3.13.4 Examples

The following sample command line disables the c0 console from the simulation:

```
mysim console disable c0
```

3.13.5 Related Commands

- **{machine} console create** on page 65
- **{machine} console enable** on page 70

3.14 {machine} console display buffered

The *{machine}* **console display buffered** command displays the status of console line buffering.

3.14.1 Tcl Syntax

machine console display buffered

3.14.2 Description

The *{machine}* **console display buffered** command displays whether or not line buffering is currently enabled in the system. The simulator is able to buffer output until an end-of-line character (**\n**) is generated. If two or more multichip modules (MCMs) are outputting to the save console, buffering allows the console lines from the different MCMs to be separated. In contrast, if buffering is disabled, the characters output by the various MCMs are interleaved by time.

3.14.3 Examples

The following command line displays whether buffering is enabled in the simulation environment:

```
mysim console display buffered
```

Displays the following output:

```
on
```

3.14.4 Related Commands

- **{machine} console create** on page 65
- **{machine} console set display buffered** on page 72

3.15 {machine} console enable

The *{machine}* **console enable** command enables a simulation console.

3.15.1 Tcl Syntax

```
machine console enable console_id
```

3.15.2 Description

An enabled input console provides input characters to the simulation; likewise, an enabled output console accepts output characters from the machine. The *{machine}* **console enable** command renders the specified console functional for input and output.

3.15.3 Arguments

console_id Specifies a name of the console to be enabled.

3.15.4 Examples

The following sample command line enables the c0 console in the simulation:

```
mysim console enable c0
```

3.15.5 Related Commands

- **{machine} console create** on page 65
- **{machine} console disable** on page 68

3.16 {machine} console list

The *{machine}* **console list** command lists information about currently defined consoles.

3.16.1 Tcl Syntax

machine console list

3.16.2 Description

The *{machine}* **console list** command lists all currently defined consoles by their respective console identifiers. For each console, the command output indicates:

- Whether the console is enabled or disabled
- Whether it is used for inputting characters, outputting characters, or both
- The console type (**file**, **listen**, **program**, **socket**, or **string**)
- The parameters for the listed console

3.16.3 Examples

The following sample lines of code demonstrate command output that is generated when the following *{machine}* **console create** commands are issued in a simulation:

```
mysim console create c0 in program xterm
mysim console create c1 in string "ls\nexit"
mysim console create outfile out file /tmp/saved_output
mysim console create input in string "callthru exit"
```

Once these consoles are created, enabled, and disabled, the *{machine}* **console list** command will display the following output:

```
c0 :: DISABLE input program : xterm
c1 :: ENABLE input string : ls\nexit
outfile :: DISABLE output file :
/tmp/saved_output input :: ENABLE input string :
callthru exit
```

3.16.4 Related Commands

- **{machine} console create** on page 65
- **{machine} console destroy** on page 67
- **{machine} console disable** on page 68
- **{machine} console enable** on page 70

3.17 {machine} console set display buffered

The *{machine}* **console set display buffered** command turns the buffered display on or off.

3.17.1 Tcl Syntax

```
machine console set display buffered on | off
```

3.17.2 Description

The *{machine}* **console set display buffered** command turns on or off a display that indicates whether or not line buffering is currently enabled in the system. The simulator is able to buffer output until an end-of-line character (**\n**) is generated. If two or more multichip modules (MCMs) are outputting to the save console, buffering allows the console lines from the different MCMs to be separated. In contrast, if buffering is disabled, the characters output by the various MCMs are interleaved by time.

3.17.3 Arguments

<i>on</i>	Turns the buffered display on.
<i>off</i>	Turns the buffered display off.

3.17.4 Examples

1. The following command line turns the buffered display on:

```
mysim console set display buffered on
```

This command displays the following output:

```
on
```

2. The following command line turns the buffered display off:

```
mysim console set display buffered off
```

This command displays the following output:

```
off
```

3.17.5 Related Commands

- **{machine} console create** on page 65
- **{machine} console display buffered** on page 69

3.18 {machine} cpu

The *{machine}* **cpu** command enables simulator commands to be called on the specified CPU.

3.18.1 Tcl Syntax

```
machine cpu cpu_number simulator_command
```

3.18.2 Description

The POWER8 Functional Simulator is designed to simulate a system in which multiple, separate processors are running on an MCM. The *{machine}* **cpu** command is a wrapper command that runs a subset of simulator commands on the processor that is specified by *cpu_number*. To view a complete list of processor-related commands, type *{machine}* **cpu** at the simulator command line.

For example, the *{machine}* **cpu** *cpu_number* **memory fread** command enables users to copy the specified number of bytes from the specified input source into the given memory address.

3.18.3 Arguments

<i>cpu_number</i>	Specifies the name of the processor on which the command operation (defined by the <i>simulator_command</i> input) is performed.
<i>simulator_command</i>	Specifies the command input that is run on the specified processor.

3.18.4 Examples

On CPU 0, read a specified number of bytes from a specified file to the specified location in memory:

```
mysim cpu 0 memory fread 0x100 1024 bootfile
```

3.18.5 Related Commands

- *{machine}* **mcm** on page 97
- *{machine}* **thread** on page 111

3.19 {machine} cycle

The *{machine} cycle* command advances the simulator by a specified number of cycles.

3.19.1 Tcl Syntax

machine cycle number_of_cycles

3.19.2 Description

The *{machine} cycle* command advances the simulated machine by a specified number of cycles. This command is useful for quickly forwarding a simulation to a specific point at which relevant metrics can be gathered for performance analysis and correlation.

3.19.3 Arguments

number_of_cycles Specifies the number of cycles to advance the machine.

3.19.4 Examples

The following sample code illustrates example Tcl steps used to advance a simulation in fast mode to the point at which the boot process is complete. At this time, the fast mode is turned off, the simulator is advanced by a specified number of cycles, and a trace generator procedure is started to verify the flow of logic or identify bottlenecks within an application:

```
...
# run through boot instructions in fast mode
mysim modify fast on

# advance to specific point in boot process
mysim cycle 160000000

# advance to specific point in boot process
mysim modify fast off

# start trace generator
proc start_emitter_readers { simulator } {
...

```

3.19.5 Related Commands

- *{machine} step* on page 109
- *{machine} tick* on page 112

3.20 {*machine*} display cycles

The *{machine} display cycles* command displays the current cycle count.

3.20.1 Tcl Syntax

machine display cycles

3.20.2 Description

The *{machine} display cycles* command displays the current cycle count. Output from this command is easily captured and directly passed as input in Tcl scripts to track the number of cycles that an application or process has run.

3.20.3 Examples

Display the cycle count after advancing the simulator 10 cycles with the *{machine} cycle* command:

```
mysim cycle 10
```

The simulator advances by the specified number of cycles, after which the *{machine} display cycles* command can be used to view how many cycles have elapsed:

```
mysim display cycles
```

The simulator summarizes the count as follows:

```
10
```

3.20.4 Related Commands

- *{machine} cycle* on page 74

3.21 {*machine*} display features

The *{machine}* **display features** command displays common features enabled in the simulation system.

3.21.1 Tcl Syntax

```
machine display features
```

3.21.2 Description

The *{machine}* **display features** command displays a summary of features that are enabled in the simulation system. The following features are available:

FLOATING_POINT Indicates that floating-point instructions are enabled.

VMX Indicates that the VMX instruction set is supported.

3.21.3 Examples

Check which features are enabled for the **mysim** simulation machine:

```
mysim display features
```

The following features are displayed for this machine:

```
FLOATING_POINT VMX
```

3.22 {machine} display fpr, fpr_as_fp, fprs

The *{machine}* **display fpr**, **display fpr_as_fp**, and **display fprs** commands display either the contents of a specified floating-point register or the contents of all floating-point registers

3.22.1 Tcl Syntax

```
machine display fpr number  
machine display fpr_as_fp number  
machine display fprs
```

3.22.2 Description

The *{machine}* **display fpr** command displays the contents of the specified floating-point register in hexadecimal format. The **display fprs** command displays the contents of all the floating-point registers in hexadecimal format. The **display fpr_as_fp** command displays the contents of the specified floating-point register in scientific floating-point format. That is:

```
systemsim % mysim cpu 0:0 set fpr 0 0x44400000200  
0x0000044400000200  
systemsim % mysim cpu 0:0 display fpr 0  
0x0000044400000200  
systemsim % mysim cpu 0:0 display fpr_as_fp 0  
2.3172194E-311
```

3.22.3 Arguments

number Identifies the floating-point register whose contents are to be displayed.

3.22.4 Examples

1. Display the contents of FPR 1 as a hexadecimal number:

```
systemsim % mysim display fpr 1
```

The following content is displayed:

```
0x0000000000000000
```

2. Display the contents of FPR 1 in scientific floating-point format:

```
systemsim % mysim display fpr_as_fp 1
```

The following content is displayed:

```
0
```

3. Display the contents of all floating-point registers as hexadecimal numbers:

```
systemsim % mysim display fprs
```

The following content is displayed:

```
0 0000000000000000
1 0000000000000000
2 0000000000000000
3 0000000000000000
4 0000000000000000
5 0000000000000000
6 0000000000000000
7 0000000000000000
8 0000000000000000
9 0000000000000000
10 0000000000000000
11 0000000000000000
12 0000000000000000
13 0000000000000000
14 0000000000000000
15 0000000000000000
16 0000000000000000
17 0000000000000000
18 0000000000000000
19 0000000000000000
20 0000000000000000
21 0000000000000000
22 0000000000000000
23 0000000000000000
24 0000000000000000
25 0000000000000000
26 0000000000000000
27 0000000000000000
28 0000000000000000
29 0000000000000000
30 0000000000000000
31 0000000000000000
```

3.23 {*machine*} display gpr, gprs

The *{machine}* **display gpr** and **display gprs** commands display either the contents of a specified general-purpose register (GPR) or the contents of all general-purpose registers. If you assigned an alias to a GPR, you can use that alias in this command instead of the number of the register.

3.23.1 Tcl Syntax

```
machine display gpr number | value  
machine display gprs
```

3.23.2 Description

The *{machine}* **display gpr** and **display gprs** commands display the contents of the specified general-purpose register or the contents of all the general-purpose registers.

3.23.3 Arguments

<i>number</i>	Identifies the general-purpose register whose contents are to be displayed.
<i>value</i>	A previously assigned alias that identifies the general-purpose register whose contents are to be displayed.

3.23.4 Examples

1. Display the contents of GPR 5:

```
systemsim % mysim display gpr 5
```

The following content is displayed:

```
0x0000000000005000
```

2. If you previously assigned the alias “count_reg” to GPR 5, you can also use the following command to display the contents of GPR5:

```
systemsim % mysim display gpr count_reg
```

The following content is displayed:

```
0x0000000000005000
```

3. Display the contents of all general-purpose registers:

```
systemsim % mysim display gprs
```

The following content is displayed:

```
0 0000000000000000  
1 0000000000000000  
2 0000000000000000  
3 0000000000000000  
4 0000000000000000
```

```
5 0000000000000000
6 0000000000000000
7 0000000000000000
8 0000000000000000
9 0000000000000000
10 0000000000000000
11 0000000000000000
12 0000000000000000
13 0000000000000000
14 0000000000000000
15 0000000000000000
16 0000000000000000
17 0000000000000000
18 0000000000000000
19 0000000000000000
20 0000000000000000
21 0000000000000000
22 0000000000000000
23 0000000000000000
24 0000000000000000
25 0000000000000000
26 0000000000000000
27 0000000000000000
28 0000000000000000
29 0000000000000000
30 0000000000000000
31 0000000000000000
```


3.24 {machine} display instruction_count

The *{machine}* **display instruction_count** command displays the current instruction count.

3.24.1 Tcl Syntax

```
machine display instruction_count
```

3.24.2 Description

The *{machine}* **display instruction_count** displays the current instruction count. Output from this command is easily captured and directly passed as input in Tcl scripts to track the number of instructions that an application or process has run.

3.24.3 Examples

Display the instruction count after advancing the simulator 1000 cycles with the *{machine}* **cycle** command:

```
mysim cycle 1000
```

The simulator advances by the specified number of cycles, after which the *{machine}* **display instruction_count** command can be used to view how many instructions have elapsed:

```
mysim display instruction_count
```

The simulator displays the count as follows:

```
1001
```

3.24.4 Related Commands

- **{machine} cycle** on page 74
- **{machine} display cycles** on page 75

3.25 {machine} display memory_size

The *{machine}* **display memory_size** command returns the memory size used for the current configuration.

3.25.1 Tcl Syntax

```
machine display memory_size
```

3.25.2 Description

The *{machine}* **display memory_size** command returns the memory size used for the current configuration. The default size is 0x0000000040000000.

3.25.3 Examples

Display the memory size:

```
mysim display memory_size
```

The simulator returns the size as follows:

```
0x0000000040000000
```

3.25.4 Related Commands

- **{machine} display cycles** on page 75

3.26 {machine} display memorymap

The *{machine}* **display memorymap** command displays information about the memory map configuration.

3.26.1 Tcl Syntax

```
machine display memorymap ?format_type?
```

3.26.2 Description

The *{machine}* **display memorymap** command displays a memory map that contains information about the start and end addresses for the memory, ROM, and PIC components.

3.26.3 Arguments

format_type (Optional) Specifies the format type for listing memory map information. The following *format_type* options are available:

<i>active:</i>	Generates command output in tabular format where information about the start and end addresses for memory, ROM, and PIC resources is listed. If a <i>format_type</i> is not specified with the command, the <i>active</i> option is the default display type.
<i>listformat:</i>	Presents the command output in a format that can be easily captured and directly passed as input in Tcl scripts.

3.26.4 Examples

1. View a memory map for the simulation in the default format:

```
mysim display memorymap
```

The simulator displays the mapping, as follows:

```
NAME:      MEMORY START: 0x0000000000000000 END:0x000000003FFFFFFF
NAME:      ROM START: 0x00000000F0000000 END:0x00000000F000007F
NAME:      PIC START: 0x00000000FFC00000 END:0x00000000FFC3FFFF
```

2. View a memory map for the simulation in list format:

```
mysim display memorymap listformat
```

The simulator displays the mapping, as follows:

```
{NAME {MEMORY} START 0x0000000000000000 END 0x000000003FFFFFFF} { {ROM} START
0x00000000F0000000 END 0x00000000F000007F} { {PIC} START 0x00000000FFC00000 END
0x00000000FFC3FFFF}
```

3.26.5 Related Commands

- **{machine} memory display** on page 98

3.27 {*machine*} display nfpr, ngpr, mode, name

The *{machine}* **display nfpr**, **display ngpr**, **display mode**, and **display name** commands are useful for tcl control scripts. The information returned contains the number of floating-point (nfpr) or general-purpose (ngpr) registers in the model, what mode (simple or turbo), and what name is in use for the configured machine.

3.27.1 Tcl Syntax

```
machine display nfpr  
machine display ngpr  
machine display mode  
machine display name
```

3.27.2 Description

The *{machine}* **display nfpr** command displays the number of floating-point registers. The **display ngpr command** displays the number of general-purpose registers. The **display mode** command indicates whether the configured machine is operating in the simple or turbo mode. The **display name** command indicates what name is used for the configured machine.

3.27.3 Examples

1. Display the number of floating-point registers:

```
systemsim % mysim display nfpr
```

The following content is displayed:

```
32
```

2. Display the number of general-purpose registers:

```
systemsim % mysim display ngpr
```

The following content is displayed:

```
32
```

3. Display the mode of the configured machine:

```
systemsim % mysim display mode
```

The following content is displayed:

```
Simulator is in mode SIMPLE
```

4. Display the name of the configured machine:

```
systemsim % mysim display name
```

The following content is displayed:

```
machine name is mysim
```

3.28 {*machine*} display number_of_MCMS

The *{machine} display number_of_MCMS* command displays the number of MCMS in the system.

3.28.1 Tcl Syntax

```
machine display number_of_MCMS
```

3.28.2 Description

The *{machine} display number_of_MCMS* command displays an integer that represents the number of MCMS that are currently configured in the simulation. The output of this command can be easily captured and directly passed as input in Tcl scripts.

3.28.3 Examples

Display the number of MCMS in the simulation system:

```
mysim display number_of_MCMS
```

The following output is displayed:

```
1
```

3.29 {machine} display slb, spr, tm, vmx, vmxr, vsxr

The **display slb**, **display spr**, **display vmx**, **display vmxr**, and **display vsxr** commands can be used to display segment-lookaside buffer (slb) translations, special purpose register (spr) values, and vector registers. For vector registers, you can display the information in vector format; you must specify the register number and the number of bytes per vector. You can also display vector registers in the full 128-bit register format.

Note: The **display_tm** command is not currently supported.

3.29.1 Tcl Syntax

```
machine display slb number | valid | all
machine display spr name | list | values
machine display tm name | list (Not currently supported)
machine display vmx reg-num size
machine display vmxr reg-num
machine display vsxr reg-num
```

3.29.2 Description

The {*machine*} **display slb**, **display spr**, **display vmx**, **display vmxr**, and **display vsxr** commands display the contents of the specified slb, spr, and vector registers.

3.29.3 Arguments

<i>number</i>	Identifies the SLB register whose contents are to be displayed.
<i>valid</i>	Indicates that only SLB entries that contain valid virtual-to-effective address mappings are to be displayed. An SLB entry is valid if the V bit is set.
<i>all</i>	Indicates that all the SLB registers are to be displayed.
<i>name</i>	Specifies the name of the SPR whose contents are to be displayed.
<i>list</i>	Requests a list of the names of all the SPRs supported in the model. Note that there are over 1,000 SPRs for the POWER8 model.
<i>values</i>	Requests a list of every SPR supported in the model and the current contents of each SPR.
<i>reg-num</i>	Specifies the register number of the vector register.
<i>size</i>	Specifies the number of bytes per vector.

3.29.4 Examples

1. Display the contents of every valid SLB register.

```
systemsimg % mysimg display slb valid
```

The following content is displayed:

#	VSID	ESID	Ks	Kp	N	L	C	V	LP	B
0	0x0000408F92C94000	0xC000000000000000	0	1	0	1	0	1	0	0
1	0x0000F09B89AF5000	0xD000000000000000	0	1	0	0	0	1	0	0
9	0x000050B3777B9000	0xF000000000000000	0	1	0	0	0	1	0	0
19	0x0000B986C02FD000	0x0000000000000000	1	1	0	0	1	1	0	0
20	0x0000C4BE37E92000	0x00000000F0000000	1	1	0	0	1	1	0	0
21	0x0000BC8479B69000	0x0000000040000000	1	1	0	0	1	1	0	0
22	0x0000BA462E918000	0x0000000010000000	1	1	0	0	1	1	0	0
23	0x0000272901DA2000	0x00000FFF00000000	1	1	0	0	1	1	0	0

2. Display the SPR named "pc."

```
systemsim % mysim display spr pc
```

The following content is displayed:

```
0x0000000000000100
```

3. Display VMX vector register 0 in vector format. There are 8 bytes per register.

```
systemsim % mysim display vmx 0 8
```

The following content is displayed:

```
0x0000000000004500, 0x000000000987600
```

4. Display VMX vector register 0 in the full 128-bit format.

```
systemsim % mysim display vmxr 0
```

The following content is displayed:

```
0x0000000000004500000000000987600
```

5. Display VSX vector register 2.

```
systemsim % mysim display vsxr 2
```

The following content is displayed:

```
0x00000000000000034560001000000
```

3.30 {machine} dtranslate

The *{machine}* **dtranslate** command translates addresses for data loads and stores.

3.30.1 Tcl Syntax

```
machine dtranslate address
```

3.30.2 Description

The *{machine}* **dtranslate** command translates an effective address to a real address for data loads and stores. This command is only useful if address translation is active. In addition, the value used as the *address* argument must be a valid EA.

3.30.3 Arguments

address Specifies the address to be translated.

3.30.4 Examples

The following *{machine}* **dtranslate** command translates the 0xC000000000011B20 effective address to the corresponding real address:

```
mysim dtranslate 0xC000000000011B20
```

The simulator displays the following result for this translation:

```
0x0000000000011B20
```

3.30.5 Related Commands

{machine} **itranslate** on page 92

3.31 {*machine*} exit

The *{machine}* **exit** command removes a machine from the simulation environment.

3.31.1 Tcl Syntax

machine exit

3.31.2 Description

The *{machine}* **exit** command permanently removes the specified simulation machine from the simulation environment and returns to the **systemsim %** command-line prompt. The *{machine}* **exit** command performs the same operation as the *{machine}* **quit** command. The **define machine** command can be used to create a new simulation machine.

3.31.3 Examples

Remove the **mysim** machine from the simulation environment:

```
mysim exit
```

3.31.4 Related Commands

- **define machine** on page 44
- *{machine}* **quit** on page 105

3.32 {*machine*} go

The *{machine}* go command starts running a machine in simulation.

3.32.1 Tcl Syntax

machine go

3.32.2 Description

The *{machine}* go command causes the simulated machine to advance its state indefinitely. The simulation continues until a stop condition is reached, which can include any of the following:

- Typing **CTRL+C** to interrupt a running simulation. The POWER8 Functional Simulator halts the simulation and returns to the **systemsim %** command-line prompt.
- An error condition that is encountered during simulation
- No processor activity occurs for three seconds after advancing the machine.

3.32.3 Examples

The following command starts the **mysim** machine:

```
mysim go
```

3.33 {machine} interrupt

The *{machine} interrupt* command schedules an interrupt of a given interrupt type.

3.33.1 Tcl Syntax

```
machine interrupt interrupt_type
```

3.33.2 Description

The *{machine} interrupt* command forces an exception (typically, a synchronous interrupts) of the given type to be raised. When the simulator subsequently is run, it performs actions defined by the core architecture to service the specified exception (that is, save machine state in appropriate registers, vector to the associated exception handler code location, and so on). This command is intended as a convenient mechanism to raise exceptions artificially, for the purposes of debugging exception handlers.

3.33.3 Arguments

interrupt_type Specifies the type of interrupt to be performed during a simulation. Many types of exceptions can be scheduled, including:

<i>AlignmentException</i>	<i>InstStorageG1</i>
<i>DataStoragePageFault</i>	<i>IPI</i>
<i>DataStorageProtection</i>	<i>MachineCheck</i>
<i>DataStorageReservationWithWriteThrough</i>	<i>MER</i>
<i>DataStorageSegmentFault dar_value</i>	<i>PerfMonitor</i>
<i>Decrementer</i>	<i>PreciseMachineCheck</i>
<i>External (msi raise lower)</i>	<i>PrivilegedInstruction</i>
<i>FPUUnavailable</i>	<i>SystemCall</i>
<i>HMI</i>	<i>SystemError</i>
<i>HV_Decrementer</i>	<i>SystemReset</i>
<i>HvSystemCall</i>	<i>TrapInstruction</i>
<i>IllegalInstruction</i>	<i>VMX_Assist</i>
	<i>VMXUnavailable</i>

If the **interrupt** command is issued without the *interrupt_type* argument, it lists all available exception types.

3.33.4 Examples

Force a load address alignment exception. The simulator is run afterwards, in order for the exception handling to take place:

```
mysim interrupt AlignmentException Load
```

3.34 {*machine*} itranslate

The {*machine*} **itranslate** command translates instruction addresses from an effective address to a real address.

3.34.1 Tcl Syntax

machine itranslate *address*

3.34.2 Description

The {*machine*} **itranslate** command translates an effective address to a real address for instructions. This command is only useful if address translation is active. In addition, the value used as the *address* argument must be a valid EA.

3.34.3 Arguments

address Specifies the address to be translated.

3.34.4 Examples

The following {*machine*} **itranslate** command translates the 0xC00000000011B20 effective address to the corresponding real address:

```
mysim itranslate 0xC00000000011B20
```

The simulator displays the following result for this translation:

```
0x000000000011B20
```

3.34.5 Related Commands

- {*machine*} **dtranslate** on page 88

3.35 {machine} load elf

The *{machine}* **load elf** command loads an ELF file into simulation memory.

3.35.1 Tcl Syntax

```
machine load elf filename
```

3.35.2 Description

The *{machine}* **load elf** command loads a properly formatted Executable and Linking Format (ELF) file into the memory of the simulated machine. Executing this command sets the initial PC and stack pointer for the program, and turns on **cpu 0**.

3.35.3 Arguments

filename Specifies the name of the ELF file to load.

3.35.4 Examples

Load the **myprog.elf** executable into memory:

```
mysim load elf /tmp/myprog.elf
```

3.35.5 Related Commands

- **{machine} load linux** on page 94
- **{machine} load vmlinux** on page 95
- **{machine} load xcoff** on page 96

3.36 {machine} load linux

The *{machine}* **load linux** command loads a Linux image into simulation memory.

3.36.1 Tcl Syntax

```
machine load linux filename
```

3.36.2 Description

The *{machine}* **load linux** command loads a properly created Linux boot image into the memory of the simulated machine. Executing this command sets the initial PC and stack pointer for the program, and turns on **cpu 0**.

3.36.3 Arguments

filename Specifies the name of the Linux image to load.

3.36.4 Examples

Load the **zImage.initrd.treeboot Linux** image into memory:

```
mysim load linux zImage.initrd.treeboot
```

3.36.5 Related Commands

- **{machine} load elf** on page 93
- **{machine} load vmlinux** on page 95
- **{machine} load xcoff** on page 96

3.37 {machine} load vmlinux

The *{machine}* **load vmlinux** command loads a vmlinux image into simulation memory.

3.37.1 Tcl Syntax

```
machine load vmlinux filename address
```

3.37.2 Description

The *{machine}* **load vmlinux** command loads a Linux kernel image into the memory of the machine. The vmlinux kernel is a modified Linux kernel that has been developed to simulate the process of loading and transferring control to the operating system kernel software. The vmlinux kernel essentially acts as a boot loader (**lilo/grub**) in a regular system. The *{machine}* **load vmlinux** command loads the modified kernel image into the memory of the simulated machine. Executing this command sets the initial PC and stack pointer for the program, and turns on **cpu 0**.

3.37.3 Arguments

<i>filename</i>	Specifies the name of the vmlinux file to load.
<i>address</i>	Specifies the physical address in memory where the image should be loaded.

3.37.4 Examples

Load the **vmlinux_2.6.7** kernel image file into memory address **0**:

```
mysim load vmlinux $IMAGE_PATH/vmlinux_2.6.7 0
```

3.37.5 Related Commands

- **{machine} load elf** on page 93
- **{machine} load linux** on page 94
- **{machine} load xcoff** on page 96

3.38 {machine} load xcoff

The *{machine}* **load xcoff** command loads the contents of an XCOFF file into simulation memory.

3.38.1 Tcl Syntax

```
machine load xcoff filename
```

3.38.2 Description

The *{machine}* **load xcoff** command loads a properly formatted XCOFF file into the memory of the simulated machine. Executing this command sets the initial PC and stack pointer for the program, and turns on **cpu 0**.

3.38.3 Arguments

filename Specifies the name of the **xcoff** file to load.

3.38.4 Examples

Load the **XCOFF** file into memory:

```
mysim load xcoff /tmp/myobject.x
```

3.38.5 Related Commands

- **{machine} load elf** on page 93
- **{machine} load linux** on page 94
- **{machine} load vmlinux** on page 95

3.39 {machine} mcm

The *{machine}* **mcm** command enables simulator commands to be called on the specified MCM.

3.39.1 Tcl Syntax

```
machine mcm mcm_number simulator_command
```

3.39.2 Description

The POWER8 Functional Simulator is designed to simulate a system in which more than one multichip module (MCM) can run its multiple, separate processors. The *{machine}* **mcm** command is a wrapper command that runs a subset of simulator commands on the MCM that is specified by *mcm_number*. To view a complete list of MCM-related commands, type *{machine}* **mcm** at the simulator command line.

For example, the *{machine}* **cpu** *cpu_number* **memory fread** command enables users to copy the specified number of bytes from the specified input source into the given memory address.

3.39.3 Arguments

<i>mcm_number</i>	Specifies the name of the MCM on which the command operation (defined by the <i>simulator_command</i> input) is performed.
<i>simulator_command</i>	Specifies the command input that is run on the specified MCM.

3.39.4 Examples

On cpu 0, read a specified number of bytes from a specified file to the specified location in memory.

```
mysim mcm 0 memory fread 0x100 1024 bootfile
```

3.39.5 Related Commands

- *{machine}* **cpu** on page 73
- *{machine}* **thread** on page 111

3.40 {machine} memory display

The *{machine}* **memory display** command displays memory addresses.

3.40.1 Tcl Syntax

```
machine memory display address unit_size ?repeat_count?
machine memory display address STRING
```

3.40.2 Description

The *{machine}* **memory display** command provides two display options for viewing memory resources. The first command displays a sequence of memory addresses starting at a given address based on a specified unit size. The unit size can be represented in bytes, halfwords (two bytes), words (four bytes), or doublewords (eight bytes).

The second *{machine}* **memory display** command returns the character that corresponds to a memory address. This command is useful for verifying whether the character that occurs at a given address matches the expected output: for example, when developing applications.

3.40.3 Arguments

<i>address</i>	Specifies the physical address where the display will begin.
<i>unit_size</i>	Specifies the size of each unit, where: <ul style="list-style-type: none"> 1 Displays the address as a byte. 2 Displays the address as a halfword. 4 Displays the address as a word. 8 Displays the address as a doubleword.
<i>repeat_count</i>	(Optional) Specifies the number of units to display. If a count is not specified, the repeat count is 1 by default.
<i>STRING</i>	Specifies that the command will display the character that corresponds to a memory address. The literal <i>STRING</i> option (all capital letters) must be passed.

3.40.4 Examples

1. Assume that your program previously wrote data to memory location 0x00B60C - 0x00B660. Display the sequence of memory addresses starting at 0x00B60C repeated 20 times in word format:

```
mysim memory display 0x00B60C 4 20
```

The output of this command results in the following:

```
0x28250000      0x40820038      0x7C7F1B78      0x7C9E238
0x4800070D      0x4801B3B5      0x3B000000      0x3C80000
0x60840000      0x788407C6      0x64840000      0x6084B60
0x7C84F214      0x4BFFFA9       0x4800005C      0x7C7F1B8
0x7C9E2378      0x7CBD2B78      0x7CDC3378      0x7CFB3B78
```

2. Display the characters that corresponds to the memory address. Assume that the memory location starting at 0x100 contains the hex digits 0x32333425.

```
mysim memory display 100 STRING
```

The output of this command results in the following:

```
234%
```

3. Using the same example data set that was used in example 1, display the memory address occurring after 0x00B60C in doubleword format:

```
mysim memory display 0x00B60C 8
```

The output of this command results in the following:

```
0x2825000040820038
```

4. Using the same example data set that was used in example 1 again, display the sequence of memory addresses starting at 0x00B60C repeated 18 times in byte format:

```
mysim memory display 0x00B60C 1 18
```

The output of this command results in the following:

```
0x28 0x25 0x00 0x00 0x40 0x82 0x00 0x38 0x7C 0x7F 0x1B 0x78 0x7C 0x9E 0x23 0x78 0x48 0x00
```

3.40.5 Related Commands

- {machine} memory **fread**, **freadcmp**, **freadgz** on page 100
- {machine} memory **fwrite**, **fwritecmp**, **fwritegz** on page 101
- {machine} memory **set** on page 102

3.41 {machine} memory fread, freadcmp, freadgz

The *{machine}* **memory fread**, **memory freadcmp**, and **memory freadgz** commands read directly from a file into memory.

3.41.1 Tcl Syntax

```
machine memory fread address number_of_bytes filename  
machine memory freadcmp address number_of_bytes_to_read compressed-filename  
machine memory freadgz address number_of_bytes_to_read compressed-filename
```

3.41.2 Description

The *{machine}* **memory fread** command implements the **Unix fread()** function. This command reads the specified number of bytes from the specified input source into the given memory address. See the “man” page on your Linux system for general information about **fread** command functionality.

The **freadcmp** and **freadgz** commands allow for use of compressed files.

3.41.3 Arguments

<i>address</i>	Specifies the physical address into which data from the input is read.
<i>number_of_bytes</i>	Specifies the number of bytes to read.
<i>filename</i>	Specifies the name of the input file that contains the data to read into memory.

3.41.4 Examples

Read data from the **saved_memory** input source into the 0x0 memory address:

```
mysim memory fread 0x0 0x4000 saved_memory
```

3.41.5 Related Commands

- **{machine} memory display** on page 98
- **{machine} memory fwrite, fwritecmp, fwritegz** on page 101
- **{machine} memory set** on page 102

3.42 {machine} memory fwrite, fwritecmp, fwritegz

The *{machine}* **memory fwrite**, **fwritecmp**, and **fwritegz** commands write directly from memory into an output file.

3.42.1 Tcl Syntax

```
machine memory fwrite address number_of_bytes filename append  
machine memory fwritecmp address number_of_bytes_to_read compressed-filename  
machine memory fwritegz address number_of_bytes_to_read compressed-filename
```

3.42.2 Description

The *{machine}* **memory fwrite** command implements the Unix **fwrite()** function. This command writes the specified number of bytes from the specified memory address into the output file. See the “man” page on your Linux system for general information about **fwrite** command functionality.

The **fwritecmp** and **fwritegz** commands allow for use of compressed files.

3.42.3 Arguments

<i>address</i>	Specifies the physical address from which data is read.
<i>number_of_bytes</i>	Specifies the number of bytes to read.
<i>filename</i>	Specifies the name of the output file into which data from memory is written.
<i>append</i>	Specifies that this write should be appended to the existing file.

3.42.4 Examples

Write data from the 0x0 memory address into the **saved_memory** output file:

```
mysim memory fwrite 0x0 0x4000 saved_memory
```

3.42.5 Related Commands

- **{machine} memory display** on page 98
- **{machine} memory fread, freadcmp, freadgz** on page 100
- **{machine} memory set** on page 102

3.43 {machine} memory set

The *{machine}* **memory set** command sets the memory address to a specified value.

3.43.1 Tcl Syntax

```
machine memory set address unit_size 64-bit_value
```

3.43.2 Description

The *{machine}* **memory set** command sets a small section of memory to the given value based on a specified size of memory (the unit size). The unit size can be represented in bytes, halfwords (two bytes), words (four bytes), or doublewords (eight bytes).

3.43.3 Arguments

<i>address</i>	Specifies the physical memory address to be written.
<i>unit_size</i>	Specifies the number of bytes to be written, where: <ol style="list-style-type: none">1 Represents a byte of memory.2 Represents a halfword of memory.4 Represents a word of memory.8 Represents a doubleword of memory.
<i>64-bit_value</i>	Specifies the value to be written into the memory address.

3.43.4 Examples

Set the memory address at 0x40562 to the 0x0003C00 64-bit value:

```
mysim memory set 0x40562 4 0x0003C00
```

Set the memory address at 0x40562 to the 0x00503D6020C43D40 64-bit value:

```
mysim memory set 0x40562 8 0x00503D6020C43D40
```

3.43.5 Related Commands

- **{machine} memory display** on page 98
- **{machine} memory fread, freadcmp, freadgz** on page 100
- **{machine} memory fwrite, fwritecmp, fwritegz** on page 101

3.44 {*machine*} mode

The *{machine} mode* command sets the simulator execution mode.

3.44.1 Tcl Syntax

```
machine mode mode_selection
```

3.44.2 Description

The *{machine} mode* command is used to switch from simple mode to turbo mode. In simple mode, one instruction is decoded or executed at a time. Simple mode maintains architectural correctness at the register level, which can be useful for debugging complex problems. In turbo mode, multiple instructions are decoded and dynamically converted to host system instructions. Then, the instructions are executed in a large block. Turbo mode is faster. It is typically used for booting an operating system and running application code or for fast forwarding a simulation run for millions of cycles to quickly get to the area of interest.

3.44.3 Arguments

mode_selection Specifies simple or turbo mode.

3.44.4 Examples

1. Set the simulator execution mode to simple:

```
mysim mode simple
```

2. Set the simulator execution mode to turbo:

```
mysim mode turbo
```

3.45 {*machine*} process

The *{machine}* **process** command can be used to determine the PID for the current process or to allow script control to change execution to another process.

3.45.1 Tcl Syntax

```
machine process current  
machine process switch newpid
```

3.45.2 Description

Use the *{machine}* **process current** command to determine the PID of the current process. Use the **process switch newpid** command to change to the “newpid.”

3.45.3 Examples

Determine the PID for the current process:

```
systemsim % mysim process current
```

The simulator displays following output for this command:

```
1
```


3.46 {*machine*} quit

The *{machine}* **quit** command removes a machine from the simulation environment.

3.46.1 Tcl Syntax

machine quit

3.46.2 Description

The *{machine}* **quit** command permanently removes the specified simulation machine from the simulation environment and returns to the **systemsim %** command-line prompt. The *{machine}* **quit** command performs the same operation as the *{machine}* **exit** command. The **define machine** command can be used to create a new simulation machine.

3.46.3 Examples

Remove the **mysim** machine from the simulation environment:

```
mysim quit
```

3.46.4 Related Commands

- **define machine** on page 44
- *{machine}* **exit** on page 89

3.47 {machine} setargs

The *{machine}* **setargs** command passes command-line arguments to standalone applications.

3.47.1 Tcl Syntax

```
machine setargs arguments_list
```

3.47.2 Description

The *{machine}* **setargs** command passes command-line arguments to standalone applications. The command determines the number of arguments that are being passed and the amount of address space that is needed to store the arguments. It allocates this amount of space in the data segment before the first text segment, and places the values or references in the newly allotted space. The contents of GPR[3] and GPR[4] are modified accordingly.

3.47.3 Arguments

arguments_list Enumerates the list of arguments that are passed to the application.

3.47.4 Examples

The following sample lines of code added in a Tcl setup file pass an argument to a standalone ELF image. The CPU number is specified with the *{machine}* **setargs** command:

```
.  
.   
.   
# load the application image and pass arguments  
mysim load elf $IMAGES/my_machine/standalone/elf_program.img  
mysim cpu 0 setargs 32769  
.   
.   
.   
mysim go
```

3.47.5 Related Commands

- **{machine} cpu** on page 73
- **{machine} load elf** on page 93
- **{machine} load linux** on page 94
- **{machine} load vmlinux** on page 95
- **{machine} load xcoff** on page 96

3.48 {machine} stall

The *{machine} stall* command stops the processor component.

3.48.1 Tcl Syntax

```
machine stall
```

3.48.2 Description

The *{machine} stall* command controls the state of the default processor (CPU 0) by putting the processor in the stall state. This command is the opposite of the *{machine} config_on* command operation. In hardware, this operation is generally implemented by executing a halt instruction or by moving a mode bit in the processor's Machine State Register (MSR). The *{machine} stall* command is useful for dynamically switching off a processor. For example, it can be used when a standalone application, which has been running in a multiprocessor environment, needs to validate functionality on a single-processor system. In this case, the **stall** command can be issued for the CPU that must be stopped (that is, **mysim stall cpu 1 stall** to stop CPU 1).

3.48.3 Examples

In a machine with a single processor, stall the processor:

```
mysim stall
```

The following message is displayed to confirm that the processor has stopped:

```
Thread 0:0:0 stalled  
CPU 0:0 stalled
```

3.48.4 Related Commands

- *{machine} config_on* on page 64
- *{machine} cpu* on page 73

3.49 {machine} start_thread

The *{machine}* **start_thread** command starts a thread.

3.49.1 Tcl Syntax

```
machine start_thread PC_address
```

3.49.2 Description

The *{machine}* **start_thread** command has a single parameter that enables the user to specify the starting address for thread execution.

3.49.3 Arguments

PC_address Specifies the starting address for thread execution.

3.49.4 Related Commands

- {machine} **stop_thread** on page 110

3.50 {*machine*} step

The *{machine} step* command advances the simulator by a specified number of instructions.

3.50.1 Tcl Syntax

machine step number_of_steps

3.50.2 Description

The *{machine} step* command advances the simulated machine by a specified number of instructions. Although this command is sometimes used for quickly forwarding the simulator to a specific point in the simulation, the *{machine} cycle* and *{machine} tick* commands are more commonly used to advance the system.

3.50.3 Arguments

number_of_steps Specifies the number of instructions to advance the machine.

3.50.4 Examples

Advance the simulator by 10000 steps:

```
mysim step 10000
```

3.50.5 Related Commands

- *{machine} cycle* on page 74
- *{machine} tick* on page 112

3.51 {machine} stop_thread

The *{machine}* **stop_thread** command stops a thread.

3.51.1 Tcl Syntax

machine stop_thread

3.51.2 Description

The *{machine}* **stop_thread** command enables the user to stop thread execution.

3.51.3 Related Commands

- {machine} start_thread on page 108

3.52 {machine} thread

The *{machine} thread* command enables simulator commands to be called on the specified thread.

3.52.1 Tcl Syntax

```
machine thread thread_number simulator_command
```

3.52.2 Description

The POWER8 Functional Simulator is designed to simulate a system in which one or more processors can run multiple threads on a multichip module (MCM). The *{machine} thread* command is a wrapper command that runs a subset of simulator commands on the thread that is specified by *thread_number*. To view a complete list of thread-related commands, type *{machine} thread* at the simulator command line.

For example, the *{machine} mcm mcm_number cpu cpu_number thread thread_number stall* command stalls the selected thread on the selected core and MCM.

3.52.3 Arguments

<i>thread_number</i>	Specifies the name of the thread on which the command operation (defined by the <i>simulator_command</i> input) is performed.
<i>simulator_command</i>	Specifies the command input that is run on the specified thread.

3.52.4 Examples

Change the state of thread 0 running on CPU 0, located on MCM 0 to the stall state.

```
mysim mcm 0 cpu 0 thread 0 stall
```

The simulator prints out the following message:

```
Thread 0:0:0 stalled  
CPU 0:0 stalled
```

3.52.5 Related Commands

- *{machine} cpu* on page 73
- *{machine} mcm* on page 97

3.53 {machine} tick

The *{machine} tick* command advances the simulator by a specified number of ticks.

3.53.1 Tcl Syntax

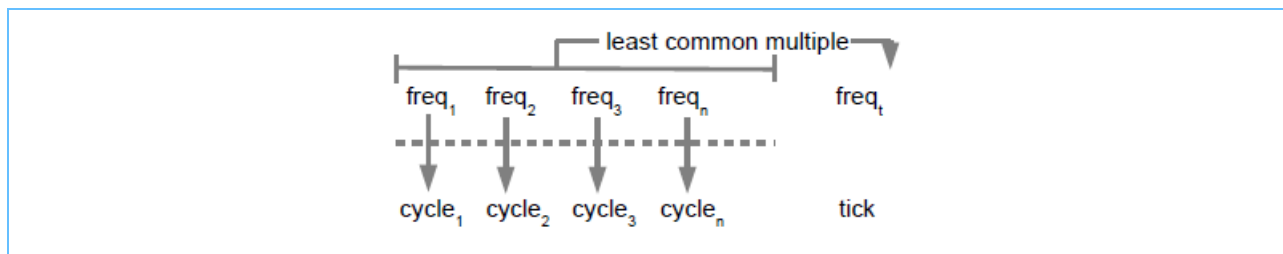
```
machine tick number_of_ticks
```

3.53.2 Description

The *{machine} tick* command advances the simulated machine by a specified number of ticks. In the simulation environment, a tick is a unifying representation of time as a function of the collective frequencies in the simulation system. In real systems, individual components probably have different perceptions of time based on their individual time domains. A time domain defines a frequency that sets the duration of a cycle. Events in a time domain are expressed in terms of its frequency cycles. In a simulation, time domains are encapsulated and declared as rational values relative to a base frequency. In this approach, each component can declare its own time domain to define its perception of time.

Figure 3-1 illustrates the relationship between the computation of a tick value and frequency cycles in the system.

Figure 3-1. Calculating Ticks in the System



To calculate the value of a tick, the simulator computes the least common multiple of the frequencies in the system. Then, for example, if a system contains n number of components whose frequencies are defined as: $freq1 = 2$ GHz, $freq2 = 4$ GHz, $freq3 = 6$ GHz, and $freqn = 12$ GHz, then one tick in this simulated system represents 1/12,000,000 of a second.

3.53.3 Arguments

number_of_ticks Specifies the number of ticks to advance the machine.

3.53.4 Examples

Advance the simulator by 10000 ticks:

```
mysim tick 10000
```

3.53.5 Related Commands

- *{machine} cycle* on page 74
- *{machine} step* on page 109

3.54 {machine} to_cycle

The *{machine} to_cycle* command advances the simulator to a specified point in the simulation.

3.54.1 Tcl Syntax

machine to_cycle number

3.54.2 Description

The *{machine} to_cycle* command advances the simulated machine to the specified point in the simulation. This command is useful for forwarding a simulation to a specific point, such as for debugging an application issue.

3.54.3 Arguments

number Specifies the point in the simulation to advance the machine.

3.54.4 Examples

The following sample code illustrates example Tcl steps used to advance a simulation in fast mode to a given point in the simulation:

```
.  
.   
.   
# advance to specific point in boot process  
mysim to_cycle 160000000  
  
# start trace generator  
proc start_emitter_readers { simulator } {  
.   
.   
.   
}
```

3.54.5 Related Commands

- *{machine} cycle* on page 74
- *{machine} step* on page 109
- *{machine} tick* on page 112

3.55 {*machine*} util dtranslate

The *{machine} util dtranslate* command translates the effective address given to a real address as a data reference.

3.55.1 Tcl Syntax

```
machine util dtranslate address
```

3.55.2 Description

The *{machine} util dtranslate* command translates the effective address given to a real address as a data reference. The translation uses the DERAT (if there is one) and DTLBs (if there are both instruction and data TLBs).

3.55.3 Arguments

address Specifies a 32-bit or 64-bit effective address to be translated.

3.55.4 Examples

Translate the 0x145772 effective address to a real address:

```
mysim util dtranslate 0x145772  
0x00000000c04567f0
```

3.55.5 Related Commands

- *{machine} util itranslate* on page 116

3.56 {machine} util dtranslate_wimg

The *{machine} util dtranslate_wimg* command translates the effective address given to a real address as a data reference. It also shows the WIMG bits associated with the address provided for translation, where:

W	Write through
I	Caching inhibited
M	Memory coherency required
G	Guarded

3.56.1 Tcl Syntax

```
machine util dtranslate_wimg address
```

3.56.2 Description

The *{machine} util dtranslate_wimg* command translates the effective address given to a real address as a data reference. The translation uses the DERAT (if there is one) and DTLBs (if there are both instruction and data TLBs). The command also shows the WIMG bits associated with the address provided for translation.

3.56.3 Arguments

address Specifies a 32-bit or 64-bit effective address to be translated.

3.56.4 Examples

Translate the 0x4328 effective address to a real address and show the WIMG bits:

```
mysim util dtranslate_wimg 0x4328
0x00
systemsim % mysim util dtranslate 0x4038
0x0000000000004038
```

3.56.5 Related Commands

- *{machine} util itranslate* on page 116

3.57 {machine} util itranslate

The *{machine}* **util itranslate** command translates the effective address given to a real address as an instruction fetch.

3.57.1 Tcl Syntax

```
machine util itranslate address
```

3.57.2 Description

The *{machine}* **util itranslate** command translates the effective address given to a real address as an instruction fetch. The translation uses the IERAT (if there is one) and ITLBs (if there are both instruction and data TLBs).

3.57.3 Arguments

address Specifies a 32-bit or 64-bit effective address to be translated.

3.57.4 Examples

Translate the 0x145772 effective address to a real address:

```
mysim util itranslate 0x145772  
0x00000000028190926
```

3.57.5 Related Commands

- **{machine} util dtranslate** on page 114

3.58 {machine} util itranslate_wimg

The *{machine} util itranslate_wimg* command translates the effective address given to a real address as an instruction fetch. It also shows the WIMG bits associated with the address provided for translation, where:

W	Write through
I	Caching inhibited
M	Memory coherency required
G	Guarded

3.58.1 Tcl Syntax

```
machine util itranslate_wimg address
```

3.58.2 Description

The *{machine} util itranslate_wimg* command translates the effective address given to a real address as an instruction fetch. The translation uses the [IERAT](#) (if there is one) and [ITLBs](#) (if there are both instruction and data TLBs). The command also shows the WIMG bits associated with the address provided for translation.

3.58.3 Arguments

address Specifies a 32-bit or 64-bit effective address to be translated.

3.58.4 Examples

Translate the 0x4328 effective address to a real address and show the WIMG bits:

```
mysim util itranslate_wimg 0x4328
0x00
systemsim % mysim util itranslate 0x4038
0x00000000000004038
```

3.58.5 Related Commands

- *{machine} util dtranslate* on page 114

3.59 {*machine*} util ppc_disasm

The {*machine*} **util ppc_disasm** command interprets an instruction as a POWER instruction executed at a given address.

3.59.1 Tcl Syntax

```
machine util ppc_disasm instruction address
```

3.59.2 Description

The {*machine*} **util ppc_disasm** command interprets the specified instruction as a POWER instruction that can be executed at the given address, and prints the assembly language interpretation of this instruction. The *address* input is required for all instructions, but used only for the interpretation of relative addresses, such as in branches.

3.59.3 Arguments

<i>instruction</i>	Specifies a 32-bit number to be interpreted as an instruction.
<i>address</i>	Specifies the address where the instruction is to be stored.

3.59.4 Examples

Interpret 0x74003100 as an instruction that is stored to the 0x4328 address:

```
mysim util ppc_disasm 0x74003100 0x4328
```

The simulator displays following output for this command:

```
andis. r0,r0,0x3100
```

3.60 {machine} util stuff

The *{machine} util stuff* command is used to stuff an instruction (a 32-bit hexadecimal value) directly into the execution stream of an active thread.

3.60.1 Tcl Syntax

machine util stuff instruction

3.60.2 Description

The *{machine} util stuff* command, which inserts an instruction into an execution stream, is designed to be used by a test execution script.

3.60.3 Arguments

instruction Specifies a 32-bit number to be interpreted as an instruction.

3.60.4 Examples

Insert the instruction 0x74003100 into an active thread:

```
mysim util stuff 0x74003100
```

3.60.5 Related Commands

- *{machine} util dtranslate* on page 114
- *{machine} util dtranslate_wimg* on page 115
- *{machine} util itranslate* on page 116
- *{machine} util itranslate_wimg* on page 117
- *{machine} util ppc_disasm* on page 118