virtutech

Simics/PPC-Simple Target Guide

Simics Version	3.0
Revision	1406
Date	2008-02-20

VIRTUTECH CONFIDENTIAL

Contents

Chapter 1

About Simics Documentation

1.1 Conventions

Let us take a quick look at the conventions used throughout the Simics documentation. Scripts, screen dumps and code fragments are presented in a `monospace` font. In screen dumps, user input is always presented in bold font, as in:

```
Welcome to the Simics prompt
simics> this is something that you should type
```

Sometimes, artificial line breaks may be introduced to prevent the text from being too wide. When such a break occurs, it is indicated by a small arrow pointing down, showing that the interrupted text continues on the next line:

```
This is an artificial ↵
line break that shouldn't be there.
```

The directory where Simics is installed is referred to as `[simics]`, for example when mentioning the `[simics]/README` file. In the same way, the shortcut `[workspace]` is used to point at the user's workspace directory.

1.2 Simics Guides and Manuals

Simics comes with several guides and manuals, which will be briefly described here. All documentation can be found in `[simics]/doc` as Windows Help files (on Windows), HTML files (on Unix) and PDF files (on both platforms). The new Eclipse-based interface also includes Simics documentation in its own help system.

Simics Installation Guide for Unix and for Windows

These guides describe how to install Simics and provide a short description of an installed Simics package. They also cover the additional steps needed for certain features of Simics to work (connection to real network, building new Simics modules, . . .).

Simics User Guide for Unix and for Windows

These guides focus on getting a new user up to speed with Simics, providing information on Simics features such as debugging, profiling, networks, machine configuration and scripting.

Simics Eclipse User Guide

This is an alternative User Guide describing Simics and its new Eclipse-based graphical user interface.

Simics Target Guides

These guides provide more specific information on the different architectures simulated by Simics and the example machines that are provided. They explain how the machine configurations are built and how they can be changed, as well as how to install new operating systems. They also list potential limitations of the models.

Simics Programming Guide

This guide explains how to extend Simics by creating new devices and new commands. It gives a broad overview of how to work with modules and how to develop new classes and objects that fit in the Simics environment. It is only available when the DML add-on package has been installed.

DML Tutorial

This tutorial will give you a gentle and practical introduction to the Device Modeling Language (DML), guiding you through the creation of a simple device. It is only available when the DML add-on package has been installed.

DML Reference Manual

This manual provides a complete reference of DML used for developing new devices with Simics. It is only available when the DML add-on package has been installed.

Simics Reference Manual

This manual provides complete information on all commands, modules, classes and haps implemented by Simics as well as the functions and data types defined in the Simics API.

Simics Micro-Architectural Interface

This guide describes the cycle-accurate extensions of Simics (Micro-Architecture Interface or MAI) and provides information on how to write your own processor timing models. It is only available when the DML add-on package has been installed.

RELEASENOTES and LIMITATIONS files

These files are located in Simics's main directory (i.e., `[simics]`). They list limitations, changes and improvements on a per-version basis. They are the best source of information on new functionalities and specific bug fixes.

Simics Technical FAQ

This document is available on the Virtutech website at `http://www.simics.net/support`. It answers many questions that come up regularly on the support forums.

Simics Support Forum

The Simics Support Forum is the main support tool for Simics. You can access it at `http://www.simics.net`.

Other Interesting Documents

Simics uses Python as its main script language. A Python tutorial is available at `http://www.python.org/doc/2.4/tut/tut.html`. The complete Python documentation is located at `http://www.python.org/doc/2.4/`.

Chapter 2

Simics/PPC-Simple Overview

Simics/PPC-Simple models a simple PPC system with a PPC603e or PPC440GP processor. Virtutech provides no images to run any OS on the machines. The machines are setup as simple as possible.

Chapter 3

Simulated Machines

Simics scripts for starting PPC-Simple machines are located in the `[workspace]/targets/ppc-simple/` directory, while the actual configuration scripts can be found in `[simics]/targets/ppc-simple/`.

3.1 PPC-Simple

The default configuration can be modified as described in section 3.2.

3.1.1 PPC-Simple Scripts

This chapter explains the files used to setup the PPC-Simple machines.

`ppc-simple-common.simics`
Starts the ppc-simple machine with the default configuration.

`ppc-simple-440gp-common.simics`
Starts the ppc-simple machine with a PPC440GP processor.

3.2 Parameters for Machine Scripts

The following parameters can be set before running the `ppc-simple-common.simics` scripts.

3.2.1 ppc-simple-common

$cpu_class
The processor class for the system.

$freq_mhz
The clock frequency in MHz for the processor.

$memory_megs
Amount of RAM in megabytes installed.

Chapter 4

Supported Components

The following sections list components that are supported for the PPC-Simple architecture. There also exist other components in Simics, such as various PCI devices, that may work for PPC-Simple but that have not been tested.

The default machines are constructed from components in the `-system.include` files in `[simics]/targets/ppc-simple/`. See the Configuration and Checkpointing chapter in the Simics User Guide for information on how to define your own machine, or make modifications to an existing machine.

4.1 PPC-Simple Components

4.1.1 ppc-simple

Description

A simple system containing a single PPC processor, some memory and a serial device.

Attributes

cpu_class

Required attribute; **read/write** access; type: **String**.

Processor type, PowerPC processor to use.

cpu_frequency

Required attribute; **read/write** access; type: **Float**.

Processor frequency in MHz.

memory_megs

Required attribute; **read/write** access; type: **Integer**.

The amount of RAM in megabytes.

Commands

create-ppc-simple [*"name"*] *"cpu_class" cpu_frequency memory_megs*

Creates a non-instantiated component of the class "ppc-simple". If *name* is not

specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

new-ppc-simple [*"name"*] *"cpu_class" cpu_frequency memory_megs*
Creates an instantiated component of the class "ppc-simple". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

<ppc-simple>.info
Print detailed information about the configuration of the device.

<ppc-simple>.status
Print detailed information about the current status of the device.

Connectors

Name	Type	Direction
uart0	serial	down

4.2 Standard Components

4.2.1 std-ethernet-link

Description
The "std-ethernet-link" component represents a standard Ethernet link.

Attributes

frame_echo
Optional attribute; **read/write** access; type: **Integer**.
Set this attribute to echo frames back to the sender. Default is not to echo frames.

link_name
Optional attribute; **read/write** access; type: **String**.
The name to use for the **ethernet-link** object. An error will be raised at instantiation time if the link cannot be given this name.

Commands

create-std-ethernet-link [*"name"*] [*"link_name"*] [*frame_echo*]
Creates a non-instantiated component of the class "std-ethernet-link". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

new-std-ethernet-link [*"name"*] [*"link_name"*] [*frame_echo*]
Creates an instantiated component of the class "std-ethernet-link". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

<std-ethernet-link>.info
Print detailed information about the configuration of the device.

<std-ethernet-link>.status
Print detailed information about the current status of the device.

Connectors

Name	Type	Direction
device	ethernet-link	any

4.2.2 std-serial-link

Description
The "std-serial-link" component represents a standard Serial link.

Commands

create-std-serial-link [*"name"*]
Creates a non-instantiated component of the class "std-serial-link". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

new-std-serial-link [*"name"*]
Creates an instantiated component of the class "std-serial-link". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

<std-serial-link>.info
Print detailed information about the configuration of the device.

<std-serial-link>.status
Print detailed information about the current status of the device.

Connectors

Name	Type	Direction
serial[0-1]	serial	any

4.2.3 std-text-console

Description

The "std-text-console" component represents a serial text console.

Attributes

bg_color

Optional attribute; **read/write** access; type: **String**.
The background color.

fg_color

Optional attribute; **read/write** access; type: **String**.
The foreground color.

height

Optional attribute; **read/write** access; type: **Integer**.
The height of the console window.

title

Optional attribute; **read/write** access; type: **String**.
The Window title.

width

Optional attribute; **read/write** access; type: **Integer**.
The width of the console window.

win32_font

Optional attribute; **read/write** access; type: **String**.
Font to use in the console on Windows host.

x11_font

Optional attribute; **read/write** access; type: **String**.
Font to use in the console when using X11 (Linux/Solaris host).

Commands

**create-std-text-console [*"name"*] [*"title"*] [*"bg_color"*] [*"fg_color"*] [*"x11_font"*] [*"win32_font"*] [*u*

Creates a non-instantiated component of the class "std-text-console". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

**new-std-text-console [*"name"*] [*"title"*] [*"bg_color"*] [*"fg_color"*] [*"x11_font"*] [*"win32_font"*] [*wi*

Creates an instantiated component of the class "std-text-console". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

<**std-text-console**>**.info**
Print detailed information about the configuration of the device.

<**std-text-console**>**.status**
Print detailed information about the current status of the device.

Connectors

Name	**Type**	**Direction**
serial	serial	up

4.2.4 std-server-console

Description
The "std-server-console" component represents a serial console accessible from the host using telnet.

Attributes

telnet_port
Required attribute; **read/write** access; type: **Integer**.
TCP/IP port to connect the telnet service of the console to.

Commands

create-std-server-console [*"name"*] *telnet_port*
Creates a non-instantiated component of the class "std-server-console". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

new-std-server-console [*"name"*] *telnet_port*
Creates an instantiated component of the class "std-server-console". If *name* is not specified, the component will get a class-specific default name. The other arguments correspond to class attributes.

<**std-server-console**>**.info**
Print detailed information about the configuration of the device.

<std-server-console>.status
Print detailed information about the current status of the device.

Connectors

Name	Type	Direction
serial	serial	up

4.3 Base Components

The base components are abstract classes that contain generic component attributes and commands available for all components.

4.3.1 component

Description
Base component class, should not be instantiated.

Attributes

connections
Optional attribute; **read/write** access; type: **[[sos]*]**.
List of connections for the component. The format is a list of lists, each containing the name of the connector, the connected component, and the name of the connector on the other component.

connectors
Pseudo class attribute; **read-only** access; type: **D**.
Dictionary of dictionaries with connectors defined by this component class, indexed by name. Each connector contains the name of the connector "type", a "direction" ("up", "down" or "any"), a flag indicating if the connector can be "empty", another flag that is set if the connector is "hotplug" capable, and finally a flag that is TRUE if muliple connections to this connector is allowed.

instantiated
Optional attribute; **read/write** access; type: **b**.
Set to TRUE if the component has been instantiated.

object_list
Optional attribute; **read/write** access; type: **D**.
Dictionary with objects that the component consists of.

object_prefix
Optional attribute; **read/write** access; type: **String**.
Object prefix string used by the component. The prefix is typically set by the **set-component-prefix** command before the component is created.

top_component

Optional attribute; **read/write** access; type: **Object**.

The top level component. Attribute is not valid until the component has been instantiated.

top_level

Optional attribute; **read/write** access; type: **b**.

Set to TRUE for top-level components, i.e. the root of a hierarchy.

4.3.2 top-component

Description

Base top-level component class, should not be instantiated.

Attributes

components

Optional attribute; **read/write** access; type: **[o∗]**.

List of components below the the top-level component. This attribute is not valid until the object has been instantiated.

cpu_list

Optional attribute; **read/write** access; type: **[o∗]**.

List of all processors below the the top-level component. This attribute is not valid until the object has been instantiated.

Chapter 5

Miscellaneous Notes

5.1 Changing the Processor Clock Frequency

The clock frequency of a simulated processor can be set arbitrarily in Simics. This will not affect the actual speed of simulation, but it will affect the number of instructions that need to be executed for a certain amount of simulated time to pass. If your execution only depends on executing a certain number of instructions, increasing the clock frequency will take the same amount of host time (but a shorter amount of target time). However, if there are time based delays of some kind in the simulation, these will take longer to execute.

At a simulated 1 MHz, one million target instructions will correspond to a simulated second (assuming the simple default timing of one cycle per instruction). At 100 MHz, on the other hand, it will take 100 million target instructions to complete a simulated second. So with a higher clock frequency, less simulated target time is going to pass for a certain period of host execution time.

If Simics is used to emulate an interactive system (especially one with a graphical user interface) it is a good idea to set the clock frequency quite low. Keyboard and mouse inputs events are handled by periodic interrupts in most operating systems, using a higher clock frequency will result in longer delays between invocations of periodic interrupts. Thus, the simulated system will feel slower in its user response, and update the mouse cursor position etc. less frequently. If this is a problem, the best technique for running experiments at a high clock frequency is to first complete the configuration of the machine using a low clock frequency. Save all configuration changes to a disk diff (like when installing operating systems). Then change the configuration to use a higher a clock frequency and reboot the target machine.

Note that for a lightly-loaded machine (for example, working at an interactive prompt on a serial console to an embedded Linux system), Simics will often execute quickly enough at the real target clock frequency that there is no need to artifically lower it.

5.2 Manually Testing Interrupts

Interrupts from the interrupt controller comes in to the Simics PowerPC via the `simple_interrupt` interface. To manually trigger an interrupt it is possible issue:

```
simics> @conf.cpu0.iface.simple_interrupt.interrupt(conf.cpu0, 0)
```

The command line triggers the interrupt towards the CPU. The seconds parameter (zero) indicates that this is a normal interrupt. Critical interrupts should use the value 1. The external interrupt will only be serviced (when continuing execution) if the MSR[EE] bit is set, enabling external interrupts. To manually set this bit issue:

```
simics> %msr = %msr | 1<<15
```

To lower the external interrupt manually issue:

```
simics> @conf.cpu0.iface.simple_interrupt.interrupt_clear(conf.cpu0, 0)
```

5.3 Cache Simulation

For generic information on how cache simulation is done in Simics please refer to Simics User Guide.

PowerPC instructions which manipulates the cache directly, such as `dcbf` can effect the cache model provided that the processor's `icache` and `dcache` are properly set.

The `icache` and `dcache` attributes should point to g-cache objects simulating instruction and data cache. For SMP configurations, the `cpu_group` attribute should point to a ppc-broadcast-bus object which will be informed about the caches the cpus uses. If the WIMG M-bit is set for a cache transaction, then memory coherency is required and the cache operation is sent down to the broadcast bus which distributes it to all known caches. For non-SMP configurations or if the M-bit is not set, the local cache is called directly.

The following operations are supported:

PowerPC operation	**Cache Operation**
dcbf (data cache block flush)	Cache_Control_Invalidate_Line
dcbst (data cache block store)	Cache_Control_Copyback_Line
dcbt (data cache block touch)	Cache_Control_Fetch_Line
dcbtst (data cache block touch for store)	Cache_Control_Fetch_Line
HID0[DCFI]	Cache_Control_Invalidate_Cache
HID0[ICFI]	Cache_Control_Invalidate_Cache

Other operations, such as locking cache lines, are not currently supported. The cache module receives the the cache operation via the cache_control interface.

Chapter 6

Limitations

6.1 PPC603e Limitations

This chapter contains the limitations that exist on the PowerPC 603e processor core. The SPRs listed do currently have no associated side-effect when either the register is read or written. In many cases this is not a problem even when code do use these registers.

The instructions implemented as no-operation (NOPs) will just execute without any side-effects at all.

6.1.1 Unsupported SPRs

None

6.1.2 Miscellaneous Processor Core Limitations

PMC: Performance Monitor Counters (PMC) are not supported.
Little endian mode setting in `MSR[LE]` is not implemented.
Floating-point estimate instructions are not bit exact.

6.1.3 Unimplemented Instructions

eciwx
ecowx

6.2 PPC440GP Limitations

This chapter contains the limitations that exist on the PowerPC 440GP processor core. The SPRs listed do currently have no associated side-effect when either the register is read or written. In many cases this is not a problem even when code do use these registers.

The instructions implemented as no-operation (NOPs) will just execute without any side-effects at all.

6.2.1 Unsupported SPRs

SPR name	Number	Description
CCR0	947	Core Configuration register 0
DCDBTRH	925	Data Cache Debug Tag Register High
DCDBTRL	924	Data Cache Debug Tag Register Low
DNV0	912	Data cache normal victim 0
DNV1	913	Data cache normal victim 1
DNV2	914	Data cache normal victim 2
DNV3	915	Data cache normal victim 3
DTV0	916	Data cache transient victim 0
DTV1	917	Data cache transient victim 1
DTV2	918	Data cache transient victim 2
DTV3	919	Data cache transient victim 3
DVC1	318	Data Value Compare 1
DVC2	319	Data Value Compare 2
DVLIM	920	Data cache victim limit
IAC3	314	Instruction Address Compare 3
IAC4	315	Instruction Address Compare 4
ICDBDR	979	Instruction Cache Debug Data reg
ICDBTRH	927	Instruction Cache Debug Tag Register High
ICDBTRL	926	Instruction Cache Debug Tag Register Low
INV0	880	Instruction cache normal victim 0
INV1	881	Instruction cache normal victim 1
INV2	882	Instruction cache normal victim 2
INV3	883	Instruction cache normal victim 3
ITV0	884	Instruction cache transient victim 0
ITV1	885	Instruction cache transient victim 1
ITV2	886	Instruction cache transient victim 2
ITV3	887	Instruction cache transient victim 3
IVLIM	921	Instruction cache victim limit
PIR	286	Processor ID
RSTCFG	923	Reset configuration
DBCR1	309	Debug Control register 1
DBCR2	310	Debug Control register 2
DBDR	1011	Debug Data Register

6.2.2 Miscellaneous Processor Core Limitations

PMC: Performance Monitor Counters (PMC) are not supported.

6.2.3 Unimplemented Instructions

dcread
icread
macchw
macchws
macchwsu
macchwu
machhw
machhws

machhwsu
machhwu
maclhw
maclhws
maclhwsu
maclhwu
mulchw
mulchwu
mulhhw
mulhhwu
nmacchhw
nmacchhws
nmacchw
nmacchws
nmaclhw
nmaclhws

6.2.4 Instructions Implemented as NOPs

dccci
icbt
iccci

Index

Virtutech, Inc.

1740 Technology Dr., suite 460
San Jose, CA 95110
USA

Phone +1 408-392-9150
Fax +1 408-608-0430

http://www.virtutech.com