virtutech

Simics/LEON2 Target Guide

Simics Version	3.0
Revision	1406
Date	2008-02-19

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/LEON2 Overview

Simics/LEON2 models a *LEON2 SPARC V8 processor* with the regular on chip devices and memory. Both the unmodified PMON firmware and RTEMS are known to work on Simics/LEON2.

Note that only a subset (though this subset is almost complete) of the LEON2 on chip devices are supported at the moment.

Chapter 3

Simulated Machines

Simics scripts for starting LEON2 machines are located in the `[workspace]/targets/leon2-simple/` directory, while the actual configuration scripts can be found in `[simics]/targets/leon2-simple/`.

3.1 LEON2 Parameters

The leon2-simple target takes a number of parameters. The parameters prefixed with `pmon` are further documented in the PMON firmware sourcecode.

$freq_mhz
The clock frequency of the processor in MHz

$prom_size
The size of the PROM area in bytes.

$has_sram
Set to TRUE if the the LEON has SRAM attached. If $has_sram is TRUE, the SDRAM will be mapped to address 0x60000000, if it is set to FALSE, the SDRAM will be mapped to address 0x40000000. Defaults to FALSE.

$sram_size
The size of the SRAM area in bytes, only used if $has_sram is set to TRUE.

$sdram_size
The size of the SDRAM area. Defaults to 0x40000000 (1 GiB)

$num_windows
The number of register windows available in the LEON2 cpu. The $num_windows must be a power of 2 and be in the range [2, 32].

$has_v8e_mac
TRUE if the UMAC and SMAC instructions are to be enabled. FALSE if they are disabled. Default is TRUE.

$has_v8_mul
TRUE if the UMUL and SMUL instructions are to be enabled. FALSE if they are disabled. Default is TRUE.

$has_v8_div
TRUE if the UDIV and SDIV instructions are to be enabled. FALSE if they are disabled. Default is TRUE.

$use_pmon_emulation
Set this to TRUE if the PMON firmware is to be emulated on boot. It defaults to TRUE. Note that if the PMON memory parameters don't match the machine configuration the same behaviour will ocurr as if the PMON firmware was compiled with the wrong parameters, e.g. the stack pointer could point outside valid memory.

$pmon_memcfg1
This value is used if the PMON emulation is used, it will be written to the Memory Configuration 1 on chip register. Defaults to 0x80000

$pmon_memcfg2
This value is used if the PMON emulation is used, it will be written to the Memory Configuration 2 on chip register. Defaults to 0x3806000

$pmon_ram_banks
The number of RAM banks. Defaults to 2.

$pmon_bank_size
Size of one RAM bank in bytes. Defaults to 0x20000000 (512 MiB)

$pmon_timer_scale
Timer scaling, this is the divisor when the timer scaler values are computed. Usually, this should remain at the default value. Defaults to 1000000.

$pmon_baud_rate
The baud rate of the serial ports, defaults to 38400.

3.2 LEON2 Scripts

This section explains the files and scripts that can be used to create and boot a simple LEON2 based target.

leon2-simple.simics
The `leon2-simple.simics` script is used to bring up a working LEON2 based target. It creates a LEON2 processor, some additional memory and optionally runs a PMON emulation script that perform the regular PMON firmware setup but without starting the serial port boot-loader. Note that the script does not load any binary. Loading a binary is performed by issuing a `set-pc (load-binary foo.srec)` command after the machine has been set up.

Chapter 4

Supported Components

The following sections list components that are supported for the LEON2 architecture. There also exist other components in Simics, such as various PCI devices, that may work for LEON2 but that have not been tested.

The default machines are constructed from components in the `-system.include` files in `[simics]/targets/leon2-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 LEON2 Components

4.2 Standard Components

4.2.1 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"*] [*w*

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

Chapter 5

Booting RTEMS

The Real-Time Executive for Multiprocessor Systems operating system, known as RTEMS, usually builds and links into a single binary file. This file usually includes the user application. This mean that booting RTEMS is a fairly simple task with Simics.

Booting RTEMS can be acomplished in several ways. One way is to run the regular PMON firmware and inject the binary in SREC format over the emulated serial port e.g. by copy-paste to the console window. This method is cumbersome, however.

A much easier and faster method is to use the PMON emulation that can be accessed through Simics's Python layer to do the initial firmware set-up, and then load RTEMS as either an ELF or an SREC file with the load-binary command. After the file has been loaded, it is possible to execute RTEMS by setting PC?to the start address provided by the binary.

RTEMS typically zeros the entire memory area during boot. In order to avoid repeating this process—which is completely unnecessary in Simics and can take some considerable time if the amount of memory is large—a checkpoint can be saved after the first boot in order to avoid repeating this excercise.

Another way is to patch the call to `memset` from `RTEMS_Malloc_Initialize` and turn the call into a `nop` instruction. It is also possible to identify when the memory-zeroing loop starts in the boot process and which register is used as the counter. As the following example shows, the register can then be patched on-the-fly:

```
# Load dhrystone binary, skipping memory zeroing
  set-pc (load-binary "rtems-dhrystone.elf")
  c 105089
  %o2 = 16
```

Chapter 6

Limitations

6.1 LEON2 Simics Features

Simics/LEON2 features most of the generic Simics features. There are, however, a number of features not yet supported by Simics/LEON2:

User decoder
It is currently not possible for the user to define or override instructions in Simics/LEON2.

User MMU
It is currently not possible for the user to override the MMU in Simics/LEON2.

Memory hierachy
It is currently not possible to connect a simulated memory hierachy to Simics/LEON2.

Symtable
The **symtable** module has not been updated to support the SPARC V8 ABI and is therefore not supported.

Haps
Many haps are not implemented. This means that the **break-hap** command does not work on these haps and that modules cannot depend on these unimplemented haps.

6.2 LEON2 Model Features

Simics/LEON2 implements the devices that are needed to boot and run RTEMS. Some devices functionality may be missing, and some devices are not implemented.

Memory controller registers
The memory controller registers have no effect and neither has the CCR. The write protection registers are unimplemented.

Timers
The LEON2 timer implementation utilize lazy evaluation of counter registers, this result in that the counter registers return incorrect values in some cases, e.g. when a timer is disabled the counter register will evaluate to the reload value.

The counters will also differ from real hardware as the LEON2 Simics model is not cycle accurate.

Watchdog
The watchdog timer is unimplemented.

UARTs
The UARTs do not support timing. They run at infinite speed even if a baud rate is specified.

Parallel I/O
The parallel I/O port is unimplemented.

Secondary Interrupt Controller
The secondary interrupt controller is currently not supported.

OpenCores Ethernet MAC
The OpenCores Ethernet MAC is unimplemented at the moment.

Chapter 7

References

`http://www.gaisler.com`
Gaisler Research, developers of the LEON2 processor.

`http://www.rtems.com`
RTEMS Real-Time Executive for Multiprocessor Systems

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