WIND RIVER

Wind River® Simics®

APPLICATION NOTE

Simics Feature List

4.6

Revision 4081
Date 2012-11-16

Corporate Headquarters
Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

Toll free (U.S.A.): 800-545-WIND
Telephone: 510-748-4100
Facsimile: 510-749-2010

For additional contact information, see the Wind River Web site:
`www.windriver.com`
For information on how to contact Customer Support, see:
`www.windriver.com/support`

Chapter 1

Simics Feature List

This document provides a short but comprehensive list of the major product characteristics and features of Simics 4.6. For more detailed descriptions, refer to the complete Simics documentation.

1.1 Product Characteristics

Supported Host Platforms ***(updated in 4.6)***

Operating system and hardware combinations that Simics runs on. The operating systems listed are the oldest ones supported. Newer OS versions are typically backward compatible.

- Linux (glibc 2.5, GTK 2.10), x86
- Linux (glibc 2.5, GTK 2.10), x86-64
- Windows XP SP1 and Windows Server 2003, x86
- Windows Vista and Windows Server 2003, x86-64

Deterministic Simulation

Simics is deterministic, forcing the exact same sequence of simulated instructions and device events to occur when rerunning the simulation. Simics can record all input, allowing exact reproduction of any execution even when asynchronous input is present since the recorded inputs are provided at the same points in time as in the original execution. A deterministic simulation greatly simplifies debugging of target systems.

Non-intrusive Inspection

Observation of the simulated system by scripts, GUI, command line and actions like checkpoint creation is non-intrusive; i.e., the simulation does not change although observation is being done.

Well-defined API

Simics exports two well-defined application programming interfaces (APIs) based on C available in C/C++. One for device modeling, the Device API and one for extending the simulator, the Simulator API. Almost all functions in the C APIs are also available

in Python, and there is a small Python-only API in addition. There is also a Simics Eclipse API providing basic control of a simulator session.

API Compatibility between Major Releases
The API remains compatible with previous Simics releases. When building modules written for an older version of Simics, the API that was used has to be specified.

ABI Compatibility within Major Releases
Minor releases (where only the third digit in the version number changes) remain binary compatibility with previous ones, eliminating the need to recompile modules when upgrading.

Modular Product Distribution
The Simics product is distributed as a single common base package, with several add-on packages. There is typically one add-on package for every modeled system architecture. Eclipse and Simics Model Builder are also distributed as separate packages. Installing support for a new architecture for example does not require the base package to be re-installed. Add-on packages and the base package can be upgraded independently of each other within the same major release.

Multi-user Install
Simics can be installed read-only in a global location in the file-system. Each user then creates one or more workspaces where user specific files, such as modules, scripts and checkpoint data, are saved. A workspace can easily be upgraded to use a new Simics version or changed back to use an older one.

Unicode Support
Simics has full support for Unicode in strings and filenames, simplifying use of the product in non-Latin character set environments.

1.2 Simics Hindsight

1.2.1 General Features

Reverse Execution
The ability to reverse time and simulate backwards. Includes the ability to run forward again with recorded input, thus recreating the exact same simulation (recording default on). Replaying can be turned off thus allowing alternate future simulation. It is possible to reverse a network of machines. The simulation can either jump to a point in time or run there, thus triggering breakpoints on the way. Reverse execution works even when distributing a simulation over several Simics processes.

Save and Restore of Checkpoints *(updated in 4.6)*
The simulation state of an entire Simics simulation can be saved to disk and restored at a later time in a host independent format. Saving a checkpoint can be done at any point in time during a simulation. Checkpoints are stored in a combination of human readable configuration files and binary files for large state (such as simulated disk or memory) and can be annotated with a clear text description. Checkpoints are

incremental, i.e. only data that has changed since the last checkpoint is saved. Only the simulated state is saved; session data such as simulator preferences and statistics is not. Reverse execution state is not saved in checkpoint either. Old checkpoints, at least four major versions back, can always be loaded into newer versions of Simics.

Save and Restore of Persistent Data

Persistent data, such as disk images and NVRAM contents that survive power-cycling, can be saved and restored at any time without the need to save the complete simulated state.

User Preferences

Users can edit and save common simulator settings, preferences.

Dynamic Machine Configuration System

Simics simulations are configured using objects and object attributes. The configuration language is used to setup machines and connect machines via networks. Configurations are host independent (except for explicit host dependencies) and can be created programmatically using Python, or be written in a special configuration file format. Although editing by hand is possible configurations are typically created by tools, such as the component system. Objects can be added to the simulation at any time, and there is support for removing objects in many device classes.

High-Level Component Configuration System

The primary configuration system uses components that represent actual replaceable hardware units, such as mother-boards, disks, keyboards, PCI cards and network links. Components support hot-plugging if applicable, and they also make sure that only valid hardware configurations are set up.

Hierarchical Configuration System

The component configuration system is hierarchical where new components can be created by connecting existing components together. This simplifies building large hierarchical systems with, for example, SOCs, boards, machines and nodes enclosing each other.

Recording of Asynchronous Input

Simics can record all input to a file, allowing replay at exactly the same point in time. Together with determinism, this can be used to rerun exactly the same simulation. When reverse execution is activated, recording is stored in memory. Asynchronous input includes network traffic, serial port data, keyboard and mouse input.

Hybrid Simulation

The Hybrid Simulation feature allows integration between Simics and an external, much more timing accurate simulation model. As the accurate model will run significantly slower than a Simics functional model, the functional model can be used for workload positioning. Once this has been done, the simulation state can be transferred (and transformed) into the accurate model, and simulation can be continued.

Integration of external timing accurate models require specific development, please contact Wind River for more information.

Scripting Languages

Simics has a built-in Python interpreter that can be used for both simple and more advanced scripting. The command line interface (CLI) also supports writing simple scripts. Scripts can be triggered by events in the simulated machine, called haps.

Graphical User Interface

Simics has a simple builtin graphical user interfaces that can be used for many common tasks, such as starting machines, loading and saving checkpoints, etc.

Command Line Interface

Simics has a powerful command line interface with on-line help, tab-completion on both commands and arguments and access to the Simics API using Python.

Eclipse Support *(updated in 4.6)*

Simics has support for using Eclipse as a simulator and debugger front end. The Simics support in Eclipse is installed as a set of plugins into any existing or new Eclipse setup. There is also a Simics Eclipse API available, described separately in this document.

Simics supports Eclipse versions 3.6 (Helios) and 3.7 (Indigo).

Quick Start Platforms *(new in 4.6 SP1)*

The Quick Start Platform (QSP) is a simple target model that includes a limited set of synthetic devices. It can be used to get a quick and easy start with Simics simulation. The set of devices provides what an OS requires to execute: interrupt controller, system controller, timers, memory, serial connection, ethernet, real-time clock, disk and flash. These devices are designed to be as simple as possible from the software's point of view. QSP also includes necessary board support packages (BSP).

Supported architectures: ARM, PPC32.

Multiple Virtual Clocks *(updated in 4.6)*

Simics supports an event driven simulation framework that allows processors with different clock frequencies, separate time events and instruction events queues. Time event queues may advance time in non-oldest first fashion; e.g., a round-robin time quantum style. Time events may synchronize all time domains before occurring. When distributing Simics over multiple processes, simulation can proceed in parallel while keeping determinism. Simics can also synchronize virtual time with foreign simulators using the *Time Synchronization Library* feature in the *Extension Builder* product.

Computer Industry Standards

A wide range of common computer standards are supported and implemented in Simics in some form. There is often generic support for common protocols that can be extended for specific devices.

Examples: AGP, ARINC 429, ATAPI, ATM, DMA, DDR, Ethernet, Fibre Channel, Flash, GBIC, I²C, IDE, IEEE 1394, IPv4, IPv6, ISA, MII, GMII, MIL-STD-1553, PCI, PCI-X, PCI Express, PCMCIA, PS/2, RapidIO, RS-232, SCSI, SystemC, USB, VGA, and VLAN.

1.2.2 Performance

Run-time Code Generation

Simics uses a run-time code generator to achieve high performance, often better than 10x compute slowdown. Collection of statistics is limited when running in this mode.

Supported architectures: ARMv5, ARMv6, MIPS32, MIPS64, PPC32, PPC64, SH4, SPARC-V8, SPARC-V9, x86, and x86-64.

Hardware-assisted Virtualization

The Simics VMP feature can improve the simulated performance for x86 and x86-64 based target systems by utilizing the hardware virtualization support found in modern Intel ® processors.

Limitation: Only supported on Linux hosts with Intel ® VT capable processors. For 64-bit targets, a 64-bit host is needed.

Hyper-simulation *(updated in 4.6)*

Hyper-simulation, also known as idle-loop-optimization, improves the performance of the simulation by doing fast forward of the virtual time when the target system is running code without side-effects. Users can adapt the hyper-simulation to match their own target code.

Restriction on Host Memory Consumption

The amount of host memory used by Simics to cache simulated memory and disk images can be limited to avoid memory related performance problems. Excess data is swapped out to files in a user specified directory.

Real-time Mode

Simics can be throttled to make sure that virtual time does not run faster than real time, or faster than some fixed ratio compared to real time.

1.2.3 Processor Features

Instruction Set Architectures *(updated in 4.6)*

Simics implements at least one processor for each of the following instruction set architectures:

ARMv5, ARMv6, H8, i8051, M68K, MIPS32, MIPS64, PPC32, PPC64, SH4, SPARC-V8, SPARC-V9, C64, C64+, X86, X86-64.

Instruction Set Extensions

Many instruction set architectures define groups of additional instructions for things like graphics, compact code, data parallel operations and Java support.

Supported extensions: AltiVec (PPC32), UltraSPARC, VIS 1, 2, 2+ (SPARC-V9), MMX, SSE, SSE2, SSE3, SSE4, VMX (x86), Thumb, NEON, TrustZone, VFP (ARM)

Unsupported extensions: Jazelle, Thumb-2 (ARM), MIPS-3D, MIPS16, MDMX (MIPS32), SPE, VSX (PPC32), 3DNow!, AVX, SSE5, AMD-V (X86)

Multi-processor and Multi-core Models
Target systems with multiple processors (SMP, AMP systems and multi-core processors) are supported.

Hyper-threaded Processor Models
Simics has the capability to model hyper-threaded processors such as POWER6 or UltraSPARC T1. The individual threads are typically implemented in Simics as a complete processor core.

Configurable Ratio Between Cycles and Instructions
The CPI (cycles per instruction) of a processor can be modified at run-time to get processor timing that is closer to real hardware for example. Both 0 and infinity are valid CPI values.

Generic Parameterized Cache
Simics includes a simple parameterized cache which can be configured to have parameters similar to a real cache in both single- and multi-processor systems. The parameters include size, associativity, number of levels, replacement policy and replacement algorithm. Simple miss penalties can be configured if the connected processor supports stalling.

Execution Trace
All instructions executed on the simulated processors can be traced by a Simics extension. An example extension, the trace module, is included that traces instructions as well as data accesses, and either prints them to the simulator console or to a binary file for off-line processing.

Instruction Fetch Mode
In instruction fetch mode, one instruction fetch is sent out to a user defined memory hierarchy or trace module for each instruction executed. Instruction fetch mode can be emulated for processors with a fixed sized instruction length by setting the cache-line size to the length of the instructions.

Supported architectures: ARMv5, ARMv6, MIPS32, MIPS64, PPC32, PPC64, SPARC-V8, SPARC-V9, x86, and x86-64.

Cache Access Instruction Fetch Mode
In cache access mode, instruction fetches for branch targets and cache-line crossings are sent to a user defined memory hierarchy or trace module. The size of cache-lines is configurable on a per processor basis.

Supported architectures: ARMv5, ARMv6, MIPS32, MIPS64, PPC32, PPC64, and SPARC-V9.

Stall on Instruction Fetches
Processors running in stall mode allow a user defined memory hierarchy to specify how long time memory accesses due to instruction fetches take.

Supported architecture: SPARC-V9.

Stall on Data Accesses
Processors running in stall mode allow a user defined memory hierarchy to specify how long time memory accesses due to data read and writes take. For instructions that generate multiple memory references, only the first access can be stalled.

Supported architectures: MIPS32, MIPS64, PPC32, PPC64, SPARC-V8, SPARC-V9, x86, and x86-64.

Instruction Profiling *(updated in 4.6)*
The instruction profiler collects an exact count of instruction executed on each virtual or physical address. Virtual address profiling requires process tracking for the target OS for separation of the counts for different virtual address spaces.

Supported architectures: ARM, H8, MIPS32, MIPS64, PPC32, PPC64, SPARC-V9, x86, and x86-64.

Magic Instructions
The software on the simulated system can communicate with the simulator by using magic instructions; i.e., instructions that are treated as NOPs on real hardware, but have a special, user defined, meaning in Simics.

1.2.4 Memory Features

Generic Address Spaces
The memory-space abstraction in Simics supports a generic 64-bit address space where memory and devices can be mapped in. Objects can be mapped at multiple addresses and be mapped overlapping with different priorities. Several common byte-swapping operations are supported for mappings. Memory spaces can be mapped in other spaces, with optional address translation being performed between them.

Dynamic Memory Mapping
Both memory and devices can be mapped and unmapped in memory-spaces at any time during the simulation. This is typically used when simulating system chipsets and host bridges that allow target software to modify the memory and device layout in the physical memory space.

Catch Accesses to Unmapped Areas
Memory areas that are not mapped by any device or memory can either be handled as on real hardware or caught by the simulator producing a warning or triggering a user callback.

Memory Access Callbacks
User callbacks can be installed for each memory space and are called on memory accesses. The timing-model is called before the access is performed and the snoop-device after. The timing-model may stall or terminate the access, while the snoop-device may change the value.

Data Access Profiling
Support for collecting exact counts of processor reads and writes to memory. The count

is per processor and on physical addresses. It can also be collected on originating instruction address. The profiler set used to count the accesses can be changed by the user.

1.2.5 Breakpoint Features

Memory Access Breakpoints
Breakpoint can be set on reads, writes and execution from memory. Any number of breakpoints can be used and they can be of any size. Breakpoints usually have a very small impact on performance.

Time Breakpoints
Breakpoints can be set on cycle or step count for each processor.

Instruction Breakpoints
Execution breakpoints can be set on instruction disassembly sub-strings and instruction bit patterns.

Control Register Breakpoints
Read and write breakpoints can be set on explicit control register accesses by processors. What processor registers that are defined to be control registers is processor dependent.

Processor Exception Breakpoints
The execution can be stopped when a processor takes a specific exception. Any number of exceptions can have breakpoint associated.

Device Access Breakpoints
Breakpoints can be set on accesses to memory and port mapped devices.

Hap/Trigger Breakpoints
Breakpoints can be set on hap (see Simulation and Simulator Triggers) occurrences, for example log output.

Console Output Breakpoints
It is possible to set breakpoints on console output. For text consoles, a string to break on is used. Graphical consoles have support for breaking on a rectangular bitmap area.

Reverse Execution Breakpoints
Breakpoints in Simics work both when running forward and when running the simulation in reverse.

1.2.6 Communication Channels

Communication links in Simics that work with multithreading and distribution. Links provide the only supported target communication between simulated machines that run in different host threads or on different host machines.

Ethernet
Generic models of an Ethernet cable, switch and hub.

I^2C
A link for I^2C communication.

Packet over SONET
A frame-based POS link.

RapidIO
A RapidIO link.

Serial
A serial link that can be connected to UARTs and consoles.

Signal
A link for single line signals.

Link Library
A link library is provided for creating new user defined link types, replacing the old generic message link.

1.2.7 Host Access

Exported Serial Line *(updated in 4.6)*
The ability to export a simulated serial port to the host software. On Linux systems, this is implemented by connecting the simulated device to a pseudo terminal (pty). On Windows, a virtual COM port on the host is used.

Telnet to Serial Port
Simulated serial ports can be accessed using standard telnet clients from a real host.

Host Serial Port
A simulated serial port can be connected to a real serial port on the host.

Simics File System
The Simics file system is a kernel module for the simulated operating system that gives the target machine access to the host file system.

Limitation: Linux targets only.

ISO file as CD-ROM and DVD-ROM Medium
The simulated CD-ROM drive can use a standard ISO 9660 file as CD/DVD-ROM medium.

Host CD-ROM
A CD in the host CD-ROM drive can be used as medium in a simulated CD-ROM.

Limitation: Not supported on Windows host.

Host Hard Disk
Hard disks and disk partitions on the host can be used as part of a simulated hard disk. The disk can be used in either read-only or read-write mode.

Graphical Display
Output from simulated graphics devices are displayed in a separate window on the host, a graphical console.

Limitation: Only one graphical console can be open at a time on Windows hosts for a simulated session.

Keyboard and Mouse
The keyboard and the mouse of the real host can be used to give input to the simulated keyboard and mouse. A stand-alone utility allows the user to create key mappings when a keyboard different from the real one is modeled.

System Panel *(new in 4.6)*
A System Panel is a graphical interface for controlling and inspecting a specific simulated machine. The panel can either be a concrete panel, which could represent the front of a computer with buttons and blinking LEDs; or it can be an abstract panel that can be, for example, used to inject network traffic or other data into a simulated system.

Host MIL-STD-1553 RT Bridge
A real MIL-STD-1553 bus controller can be used to bridge MS1553 traffic from a bus controller on a real MS1553 bus to simulated remote terminals in Simics. The connection requires a Condor PMC 1553 PCI card, while the simulated RTs can be of any kind.

Limitation: Windows hosts with a real Condor PMC 1553 only.

Host MIL-STD-1553 Bus Controller
A simulated MIL-STD-1553 bus controller can be connected to a real 1553 bus on the host. This connection requires a real Alphi PMC1553 PCI card, while the simulated card may be of any type. The limitation on the host device is due to lack of a standard driver interface to MS1553 devices; each vendor has its own driver software.

Limitation: Windows hosts with a real Alphi PMC 1553 only.

Port-forwarding/NAT
Allows the target systems to communicate with the real network using their own private IP addresses. Simics translates between external and internal IP addresses. Port-forwarding/NAT does not require administrator privileges to install or run. Both UDP and TCP protocols are supported, although some higher level protocols, such as FTP, may require special support. Some supported common protocols are Telnet, TFTP, FTP and HTTP.

See also "Simics Ethernet Networking" section.

1.2.8 Remote Interfaces

GDB Remote
Simics supports the GDB remote protocol over TCP/IP. Multiple GDB debuggers can be connected to Simics simultaneously, one to each target process. The reverse execution extensions to the protocol are supported. GDB versions 6.8, 7.0, 7.1 and 7.2 are supported.

WDB Remote
WDB is a Wind River debug protocol used to communicate between a debugger and an agent on the target system.

Limitation: Not all versions of Tornado and Wind River Workbench are supported.

Telnet Frontend
It is possible to connect to a running Simics using a standard telnet client for access to the command line.

1.2.9 Miscellaneous Features

Global Startup File
There is a global startup file that runs when Simics starts, that can be used in multi-user environments to give all users common settings such as paths to disk images. There is also a per-user (workspace) global startup file.

Merging of Checkpoints
A series of checkpoints can be merged into a single one using a standalone tool.

Limitation: On Windows, the tool has to be run from a console window (Command Prompt). Running from a Cygwin shell is not supported.

Simulation and Simulator Triggers
Simics defines haps as pre-defined conditions in the simulation that trigger the callback of one or more user-installed function. Users may also add new haps to the simulation. Haps are either simulation or simulator specific. Examples of the first kind are processor interrupt, control register write and magic instruction. Examples of the second kind are configuration object created, simulation stopped and preferences changed.

Time Server
The virtual time can be distributed to externally running tools, allowing them to run with the same virtual time as the simulated session. This is useful for timing sensitive tests, for example, when the real and simulated time differs.

Loading Binary Files into Simulated Memory
Simics can load binary files, including executables, directly into the simulated memory, without the need for any initial loader software. The formats supported are raw binary files, ELF, Motorola S-Record, PE32 and PE32+.

Compressed Disk Images
Disk images for simulated systems can be stored in a compressed random access format, called CRAFF, to save disk space on the host. The compression is transparent to the target system.

Load and Store of Memory and Disk to Host File
Parts of, or complete, memory and disk images in Simics can be saved to a file on the host file system.

Check for Available Product Updates
It is possible from the builtin GUI to check for product updates that are available for download.

Ethernet Frame Injection
Frames on the simulated Ethernet link can be inserted, removed and modified by the user.

Ethernet Bandwidth Limitation
Ethernet devices can be configured with a maximum send bandwidth to avoid overloading receiving network devices. This includes limiting the bandwidth on the real network connection, except for port-forwarding.

Limitation: Not all network devices support this feature.

1.3 Simics Analyzer

1.3.1 General Features

OS Aware Process Tracking *(updated in 4.6)*
Simics can detect when processes start and exist on the simulated machine, and keep track of what process that is currently active. Process tracking is useful for single process debugging, or when collecting statistics on a per process basis. Since knowledge about the target OS is needed, the process tracker has to be extended to support new operating systems. There is also a simple process tracker that only differentiates between user and supervisor mode that can be used if the OS is not supported. The Linux process tracker is capable of following the threads in a process. The VxWorks tracker can follow both RTPs and tasks.

Supported operating systems: Linux 2.4 and 2.6, OSE 5.3, VxWorks 5.5, 6.7, 6.8 and 6.9, QNX Neutrino 6.4, Wind River Hypervisor 1.0, 1.1, 1.2 and 1.3.

Supported architectures: ARM, PPC32, x86, x86-64.

Limitation: Not all combinations of target architecture and operating system are supported.

Symbolic Debugging *(updated in 4.6)*
Simics has built-in support for source code debugging of programs written in C/C++. The support includes variable inspection and line-number information for example. The binary format supported is ELF, and the debug formats are STABS and DWARF2, 3. It is possible to debug several binaries with different ABIs at the same time.

The new debugger in Simics 4.6, available in Eclipse, is currently a beta product functionality.

Unsupported formats: COFF, PE, OMF, DWARF1, PDB.

Supported Debugger ABIs
The debugger has to know about the Application Binary Interface (ABI) that the simulated software uses to be able to correctly debug code on the target system.

Supported ABIs: ARM, H8, M68K, MIPS o32, MIPS n64, PPC32 SysV, PPC64 ELF, SPARC32, SPARC64, x86 SysV, and C64+.

Code Coverage *(updated in 4.6)*

Simics supports collecting statement coverage on C/C++ and assembler source code for software running on the simulated target without any need to instrument the binary code.

Limitation: Coverage requires that the processor model in Simics supports instruction profiling. To get coverage on a single user program running on the target, a process tracker for the operating system must exist. The binary to do code coverage on has to be in a format that the GNU `objdump` can parse. This typically means the ELF binary format with debugging information in DWARF.

1.4 Simics Accelerator

Host Multithreading

Simics can utilize more that one host core to get improved simulation performance for target systems that can be partitioned into multiple machines, called *cells* in Simics.

Distributed Simulation

A simulation can be distributed over several Simics processes, either running on the same real host or on multiple networked hosts. A distributed simulation keeps the determinism of non-distributed simulations, and maintains a global time across all processes. Simulations can be distributed at link boundaries, such as Ethernet and serial port connections.

Page Sharing

Identical memory and disk pages, as well as some internal data related to simulated memory, can be shared between multiple target machines in the same simulation. This saves memory for large setups with several similar machines and may also improve performance.

1.5 Simics Model Builder

Supported Module Build Compilers *(updated in 4.6)*

To write user modules in DML, Simics requires a C compiler, the GNU make utility and other GNU binutils. On Windows hosts, the MinGW environment is required. Since code provided with Simics is written in C99, modifications may be needed when using a non-C99 compliant compiler such as MSVC.

Supported compilers:

- GCC 4.x on all host platforms.
- Microsoft Visual C++ .NET 2003, Visual C++ 2005, 2008 and 2010.

Limitation: A patched GCC compiler, that uses the MSVC ABI, is required to build modules on 32-bit Windows hosts. There is one such GCC provided for Simics in the MinGW-Simics package.

User Developed Devices

Users can develop their own models of devices by writing DML or Python. Devices can be written in C/C++ as well, although this not recommended.

DML

The device modeling tool DML simplifies the task of implementing simulation models at the right level of abstraction and makes it easier to develop models that support standard features such as inspection, checkpointing and reverse execution.

SystemC Bridge

The SystemC bridge enables a SystemC device to be integrated into a Simics simulation. This is done by compiling the SystemC kernel into a Simics module, together with the SystemC simulation models.

The user will have to write translators between the SystemC and Simics interfaces.

Limitation: For checkpointing to work, the SystemC model has to be adapted to use the Simics checkpoint mechanism. Also, it may not use threads or have some other implicit state. Reverse execution of SystemC models is not supported. There is no support for 64-bit Windows host.

IP-XACT *(new in 4.6)*

Simics is capable of both importing and exporting IP-XACT descriptions. IPX-ACT descriptions can be exported for existing Simics device models written in DML. IP-XACT descriptions can be imported and used as templates for DML devices.

System Panel Build Kit *(new in 4.6)*

The System Panel Build Kit enables the construction of System Panels. A System Panel is a graphical interface for controlling and inspecting a specific simulated machine.

Example Code

There is example code in DML and Python included for several common device classes such as DMA, I²C, PCI, interrupt, serial devices and timers.

Multithreaded User Extensions

Simics extensions may be written using threads, both in C/C++ and in Python. There is a global simulator lock that is used to synchronize access to the Simics core, and a few thread safe functions in the API.

Access to Python From C/C++

The Simics API allows extensions to evaluate Python code in the global Python interpreter environment and to get a return value back.

Access to C Functions from Python

User defined Simics interfaces can be called from Python code.

User-defined Commands
Users can add their own commands to the command line interface. Commands can be added dynamically at any time.

User-defined Processor Instructions
A user decoder can be connected to Simics that may override existing instructions or define new ones. User decoders are shared between all instances of a processor architecture.

1.6 Simics Extension Builder

1.6.1 General Features

User Developed Simulator Extensions
The simulator can be extended by modules written in Python or C/C++.

User Developed Processors
Users can plug in their own processor models in Simics using the Simics Processor API.

Simics Eclipse API
There is an API available in Eclipse for controlling a Simics simulation including a way to send user defined messages to a Simics session.

Simics as a Library
Allowing the Simics shared object (DLL) to be linked with another binary. This can be used to integrate Simics into another simulator or application for example.

Time Synchronization Library*(new in 4.6)*
Provides synchronization of the virtual time in Simics with the virtual time in an external simulator and also a deterministic way to exchange information between them, such as network traffic.

Hypersimulation Interface*(new in 4.6)*
An interface for adding user-written hypersimulation patterns (idle loop detectors) to the generic hypersimulation framework.

1.7 Simics Ethernet Networking

Ethernet Bridge
Simics can act as an Ethernet bridge between the real and simulated network, where the simulated machines appear to be on the real Ethernet network. Administrator privileges are required for installation. The simulated machines have to be configured with IP addresses matching the real network. To reduce the amount of network traffic that Simics has to process, a MAC translate mode exists where the simulated machines appear to share MAC address with the real host on the real network. Only IP based protocols work in this mode. Without MAC translation, the Ethernet addresses of the simulated machines have to be unique on the real network as well.

Virtual Host Interface on the Simulated Network
Simics can create a virtual interface on the real host that appears as a network device on the simulated network. Support for TAP interfaces is required on the host.

Network Services *(updated in 4.6)*
Simics Ethernet Networks includes a virtual network server which provides a number of common network services over IPv4, and in some cases IPv6, on Ethernet networks:

- BOOTP
- DHCP, DHCPv6
- DNS
- RARP
- TFTP
- FTP

User-extendibility of Port-forwarding/NAT
Simics exports an Application Layer Gateway API that can be used to add support for higher level protocols to port-forwarding/NAT. Protocols that send actual IP addresses in the protocol data typically need special support.

1.8 Licensing

Node-locked Licenses
Simics licenses can be locked to a specific host.

Floating Licenses
Simics licenses can be pooled, allowing several users to share the same limited set of licenses.

Borrowed Laptop Licenses
Laptop users can checkout floating licenses for a limited period of time without the need to contact the license server while it is used.